SZAKDOLGOZAT

Fási Gábor

2013

Pannon Egyetem

Villamosmérnöki és Információs Rendszerek Tanszék

Mérnök informatikus BSc szak

SZAKDOLGOZAT

Ülésszervezést és jegyzőkönyv-írást támogató keretrendszer készítése

Fási Gábor

Témavezető: Dulai Tibor

2013

Ide jön az eredeti vagy a fénymásolt feladatkiírás.

Nyilatkozat

Alulírott Fási Gábor diplomázó hallgató kijelentem, hogy a szakdolgozatot a Pannon Egyetem Villamosmérnöki és Információs Rendszerek tanszékén készítettem Mérnök informatikus BSc szak (BSc in Computer Engineering) megszerzése érdekében.

Kijelentem, hogy a szakdolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a szakdolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2013. május 02.

Aláírás

Alulírott Dulai Tibor témavezető kijelentem, hogy a szakdolgozatot Fási Gábor a Pannon Egyetem Villamosmérnöki és Információs Rendszerek tanszékén készítette Mérnök informatikus BSc szak (BSc in Computer Engineering) megszerzése érdekében.

Kijelentem, hogy a szakdolgozat védésre bocsátását engedélyezem.

Veszprém, 2013. május 02.

Aláírás

Köszönetnyilvánítás

Köszönöm a családomnak a sok türelmet és segítséget, amit kaptam, nélkülük ez a szakdolgozat nem készült volna el.

Köszönöm témavezetőmnek, Dulai Tibornak az elmúlt egy év során adott iránymutatását.

Végül, de nem utolsó sorban, szeretném megköszönni a szaktársaimnak a bíztatást.

TARTALMI ÖSSZEFOGLALÓ

E szakdolgozat témája egy olyan, böngészőn keresztül használható rendszer fejlesztése, amely egy adott szervezeten belül hatékonyan tudja segíteni ülések szervezését és azok jegyzőkönyveinek elkészítését.

A fejlesztést az egyik legelterjedtebb weben használatos nyelven, PHP-ban végeztem a Symfony2 keretrendszer használatával; ezzel biztosítva azt, hogy az alkalmazás a későbbiekben más által is könnyen módosítható és karbantartható legyen.

A kész rendszer képes a Google Naptárban eseményt létrehozni az üléseknek, hogy az összes meghívott résztvevő a saját napirendjében megtalálja; valamint a jegyzőkönyvek PDF formátumú exportálására, a papíralapú iktatást elősegítendő. Három fő- és egy segédmodulra bontottam, mindegy jól meghatározott feladatkörrel rendelkezik. A felülete könnyen átlátható és használható, a PDF kinézete egyszerűen testreszabható az üzemeltetők igényei alapján.

Kulcsszavak: *web, jegyzőkönyv, ülés, PHP, Symfony2, Doctrine, MySQL, Bootstrap, Google Naptár*

ABSTRACT

The topic of this thesis is to create a system, that is available via a browser and can aid an organisation in organising meetings and creating reports of their events.

Development was based on one of the most widespread languages for creating web-based systems, PHP, using the Symfony2 Framework; thus making sure the completed application can be modified and maintained by anybody.

The completed system is able to create an event in Google's Calendar so all invitees can see it in their own; and to export the reports to pdf. It was separated into three modules with clear separation of concerns. Its interface is easy to use and the layout of the PDF can be completely customised.

Keywords: *web, report, meeting, PHP, Symfony2, Doctrine, MySQL, Bootstrap, Google Calendar*

Tartalomjegyzék

1. A feladat összefoglalása

Témám egy olyan rendszer elkészítése, mely böngészőn keresztül használható, a megfelelő jogosultságú felhasználók képesek ülést hirdetni helyszínnel, időponttal, majd erre résztvevőket meghívni, akik erről automatikus értesítést kapnak. Szintén a rendszer feladata a lefolyt ülések jegyzőkönyvei elkészítésének segítése, ezek megőrzése és megfelelő feltételek esetén nyilvánossá tétele.

A rendszer fő haszonélvezője a Pannon Egyetem Hallgatói Önkormányzata lesz, de célom a lehetőség általánosítás, hogy a későbbiekben bárki könnyen és gyorsan az egyedi igényeihez tudja szabni és használatba venni.

2. Hasonló célú hazai és külföldi rendszerek

Számos jegyzőkönyvvezető rendszer érhető el már magyar nyelven is, de ezek többsége más típusúra specializálódott, mint az én rendszerem; például mérési-, verseny vagy vizsgajegyzőkönyv-készítésre.

Három darab ülésjegyzőkönyv-készítő funkcionalitással bíró rendszert találtam, melyek magyar készítésűek:

- *AC-TMTR, avagy az Albacomp Testületi Munkát Támogató Rendszer*[3]
  Önkormányzatok számára készült, honlapjuk szerint „Az AC-TMTR a teljes demokratikus döntéshozatali folyamat támogatására alkalmas az előterjesztések kezelésétől a döntések végrehajtásáig’’.
  A rendszer tudása messze több, mint jegyzőkönyvkészítés, Lotus Notes alapon működik.

- *InterMap e-FORTE*[10]
  Fő célcsoportja a polgármesteri hivatalok, felső államigazgatás és nagyvállalatok vezetése, ennek tudása is jelentősen meghaladja a jegyzőkönyvkészítést és ülésszervezést.
  Webalapú, ASP.NET nyelven készült.

- *eKÖZIG döntéstámogató rendszer*[9]
  Szintén a közigazgatás a fő célcsoport, nagyméretű és bonyolult rendszert

eredményezve.

Webalapú, ASP nyelven íródott.

A fenti rendszerek mindegyike fizetős, valamint a mögöttes technológia is – az operációs rendszer (Windows) és a futtatókörnyezet (ASP, ASP.NET) egyaránt komoly beruházást igényel a bevezetéskor.

Kutatásom során igyekeztem külföldi rendszereket is találni, elsősorban angol nyelven; azonban az angol *report*, illetve *proceedings* szavak nagyon általános jelentése miatt feldolgozhatatlan mennyiségű találatot kaptam. Azon alkalmazások, amelyekre sikerült rátalálnom, a magyar rendszerekhez hasonlóan nem ülés-, hanem egyéb speciális jegyzőkönyvek készítésére voltak csak alkalmasak.

3. A rendszer tervezése

A következőkben vázolom a rendszerem felépítését, az egyes komponensek feladatait, azok tervezésével kapcsolatos főbb megszorításokat, döntéseket.

3.1. Modulok

A rendszer három fő modulra bontható:

1. Felhasználókezelés, authentikáció és authorizáció
2. Ülések hirdetése, kezelése
3. Jegyzőkönyvírás

3.1.1. Felhasználókezelés, authentikáció és authorizáció

Az alkalmazás zárt, azaz nincs szabad regisztráció, csak az adminisztrátorok hozhatnak létre új felhasználót, illetve ők tudják a létezőket módosítani. A rendszer a kényelmes, gyors és biztonságos bejelentkezésre a Google OpenID rendszerét használja, így nem kell például jelszavak biztonságos tárolásáról gondoskodnunk. Minden HÖK tag rendelkezik egy Google Apps fiókkal.

Ennek a modulnak a feladata az OpenID bejelentkezési folyamat megvalósítása, a kapott adatok alapján a felhasználóhoz párosított jogosultsági szint ellenőrzése

a rendszer használata során. Szintén ide tartozik a felhasználók adminisztrátorok általi kezelése.

A felhasználók authentikációjára és authorizációjára a Symfony2 Security komponensét használom – ennek segítségével egyéb bejelentkezési módok utólag is kevés munkával hozzáadhatók a rendszerhez. Itt külön említést érdemel a klasszikus, felhasználónévvel (e-mail címmel) és jelszóval történő bejelentkezés, mely néhány sor konfigurációval és egy loginformot tartalmazó template-tel megoldható. Ilyen további bejelentkezési lehetőségek hozzáadásával megoldható, hogy valaki Google fiók nélkül is hozzáférjen a rendszerhez.

3.1.2. Ülések hirdetése, kezelése

Az ülések rendelkeznek fix számú adattal, mint a rövid leírásuk, időpontjuk és a helyszínük. Megfelelő jogosultsággal rendelkező felhasználók képesek a fenti adatok megadása után ülést hirdetni, meghívottakat megadni. Opcionális adat még az ülés hosszú leírása, itt lehet például megadni a tervezett napirendi pontokat; valamint lehetőség van dokumentumok feltöltésére, ha valami anyagot előzetesen tanulmányozni kell a résztvevőknek.

A rendszer a hirdetett üléseknek létrehoz egy bejegyzést a Google Naptár alkalmazásában, amelyre meghívja a résztvevőket, így az bekerül az ő naptárukba is, valamint erről értesítő levelet kapnak.

3.1.3. Jegyzőkönyvírás

A lezajlott ülésekhez készíthető jegyzőkönyv, de természetesen ülés nélkül is lehet jegyzőkönyvet létrehozni (korábbi ülések, vagy ha a szervezés valamely oknál fogva a rendszeren kívül történt).

A jegyzőkönyvek rendelkeznek ugyanazokkal az alapadatokkal, mint az ülések, valamint tetszőleges számú bejegyzéssel, melyekből három típust különböztetünk meg: napirendi pont, felszólalás valamint szavazás. Mind más adatokkal rendelkezik, a rendszernek kezelnie kell ezt és a bejegyzések egymás közti sorrendjét.

A papír alapú iktatást elősegítendő lehetőség van az elkészült jegyzőkönyvek PDF formátumú exportálására.

Szintén e modul feladata a nyilvános ülések kész jegyzőkönyveinek publikálása, bárki számára elérhetővé tétele.

3.2. Választott technológia

Feladatom tervezése során hangsúlyt fektettem arra, hogy nyílt forráskódú, ingyenes rendszerekre építsem a megoldásom. A hardverköltségtől eltekintve, amely mindenképpen jelen van, az ilyen szerverek összeállítása a lehető legolcsóbb, illetve szakemberek is bőven állnak rendelkezésre. Így esett választásom a PHP nyelvre és a Symfony2 keretrendszerre.

A választott követelményeimnek megfelelő webszerverekből az egyetem már most is rendelkezik több darabbal, és persze hozzájuk értő rendszergazdákkal is, így a rendszer majdani bevezetésének a költsége minimális.

3.2.1. PHP

A PHP ma az egyik legelterjedtebb programozási nyelv, melyet weboldalak készítésére használnak. Rengeteg kezdőknek szóló leírás van róla, minden webfejlesztő ismeri legalább az alapjait. Rengeteg, akár ingyenesen elérhető tárhelyszolgáltatás vehető igénybe, amelyek lehetővé teszik PHP-alkalmazások kiszolgálását.

A 2008-ban megjelent 5.3-as verzióval a nyelvbe bekerült a névterek támogatása, valamint a garbage collector algoritmusa is sokat javult, jelentős teljesítménynövekedést eredményezve. A feladatom minimum követelménye az 5.3.7-es verzió, ebben javítottak egy, a bcrypt modulban levő hibát, mely hibás (viszonylag könnyen törhető) jelszókódolást eredményezett.

3.2.2. Symfony2

A Symfony az egyik nagy keretrendszer a PHP világában. Első verziója 2005-ben jelent meg[1], csak ragasztó volt egy maréknyi könyvtár közt, azok együttes használatát segítendő. Folyamatosan fejlődött, olyan nagy oldalak használták, mint a Yahoo![16] és a Dailymotion[7]. Az 1.4-es ága 2009-ben jelent meg, és egészen 2012 novemberéig rendelkezett támogatással.

A Symfony2 hosszas fejlesztést követően jelent meg 2011 júliusában. Az egyik első nagy keretrendszer volt, mely a PHP 5.3 új lehetőségeit kihasználva lett

az alapoktól újraírva. Fejlődése a közösség bevonásával történt, több, mint 250 önkéntes segített be. Mostanra 600 feletti a fejlesztésben részt vevők száma.

A rendszerem fejlesztése a 2.1-es verzióban történt. A készítés utolsó hónapjaiban adták ki a 2.2-es verziót számos visszafelé nem kompatibilis változtatással, ezért nem frissítettem rá – a 2.1-es változat 2013 közepéig élvez hivatalos támogatást, így a rendszer további fejlesztése során mindenképpen megfontolandó a 2.2-re átállás.

A Symfony remekül együtt tud működni a Doctrine ORM-el[2, 8], ezzel lehetővé téve, hogy az alkalmazásunk számára észrevétlenül lecseréljük az adatbázis-szerverünk valamely más típusúra. A tábláink szerkezetét egységes módon kell megadnunk, és az illesztőfüggő kódot a Doctrine legenerálja nekünk. Szintén jó a támogatása az adatbázis-migrációkkal kapcsolatban, ha változik az adataink szerkezete, ezt nem kézzel kell lekövetnünk az éles szerveren, egyetlen, a terminálban kiadott paranccsal a Doctrine ezt megteszi nekünk.

3.2.3. Bootstrap

A Twitter által megalkotott Bootstrap[15] css keretrendszer segítségével könnyen és gyorsan hozhatunk létre egységes, jól kinéző webalkalmazásokat. Beépítve tartalmaz egyszerű, de jól használható stílusokat a webfejlesztés szinte minden területére, az alapkinézet megalkotásától az űrlapmezők formázásáig. Élénk közösség alakult ki körülötte, így a korábban nem lefedett területekre is számos megoldás létezik mostanra. A kezdetek óta kiegészült egy maréknyi javascript könyvtárral is, melyekkel egyszerűen tudunk például dialógusablakokat vagy hasonló, gyakori feladatokat megoldani.

3.3. Kritikus részek

A következőkben megemlítem a rendszer kritikus pontjait, illetve hogy milyen tervvel rendelkezem ezek megoldására.

3.3.1. OpenID authentikáció

Az OpenID[4] egy viszonylag összetett protokoll sok hibázási lehetőséggel. Szerencsére több készen elérhető megoldás is létezik, ezek közül nem egy pedig minimális

konfiguráció után kész együttműködni a Symfony2 biztonsági komponensével köszönhetően annak, hogy készítettek hozzá kiterjesztést (a Symfony terminológiában Bundle-t). Ezek közül az FpOpenIdBundle[11]-t választottam.

3.3.2. Jogosultságkezelés

A jogosultságok kiosztásában olyan rugalmasnak kell lennünk, amit egy egyszerű, felhasználókat csoportokra bontó megoldás nem tesz lehetővé. Az egyeztetések során hamar kiderült, hogy több olyan felhasználó is lesz, aki az eredeti rendszer szerint nem hirdethetne ülést (mert nem kari elnök), de ezt mégis lehetővé kell tenni számára.

A Symfony2 Security komponense nagyon finom jogosultság-szabályoztást tesz lehetővé onnantól, hogy egy oldalt valaki csak megfelelő joggal ér el, odáig, hogy egy adott adatbázis-rekord valamely mezőjéhez hozzáférhet-e.

3.3.3. Google Naptár együttműködés

Az API, amivel eseményeket lehet létrehozni és arra meghívottakat hozzáadni[13] nincs túlbonyolítva, és a Google ad PHP-hoz hivatalos kliens könyvtárat[12]. Ennek segítségével hoztam létre egy Symfony2 szolgáltatást, mellyel a rendszer bármely részéről könnyen tudok a Naptárral kapcsolatba lépni.

A rendszer a meghirdetett ülésekről létrehoz egy-egy naptárbejegyzést, és az ülés meghívottjait ott is felveszi a meghívottak közé – így ez bekerül az ő naptárukba is.

3.3.4. PDF exportálás

Webes rendszerek esetén gyakran előforduló kérés, hogy valamely generált dokumentumot lehessen PDF-ben is letölteni; így nem meglepő, hogy számos Symfony2 Bundle foglalkozik ezzel a feladattal.

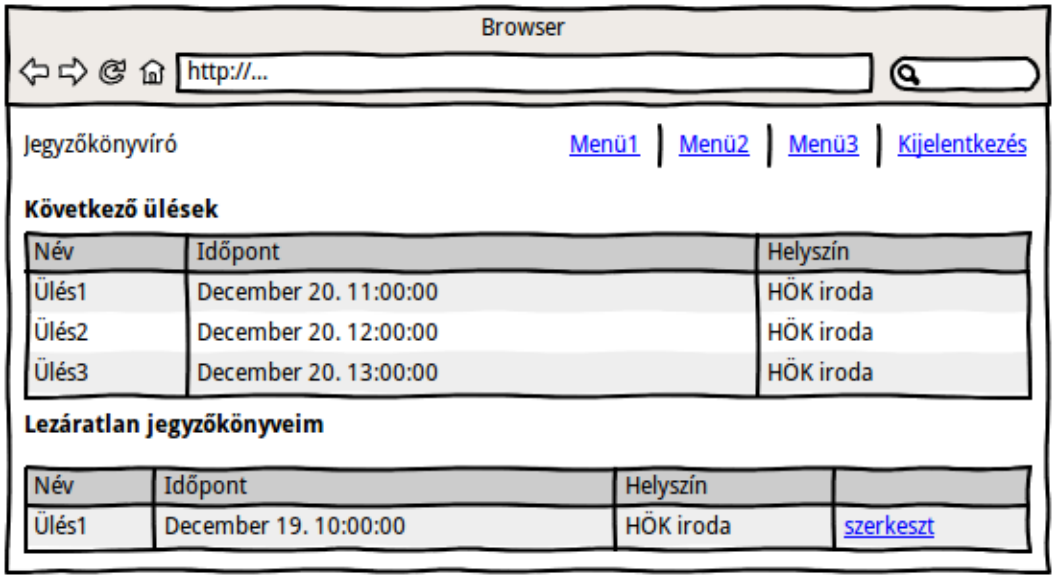
Választásom egy olyan könyvtárra esett, mely HTML bemenet alapján generálja a PDF kimenetet, így a későbbiekben a nyomtatott jegyzőkönyv kinézete, az elemek sorrendje, formátuma könnyen módosítható a felhasználók igényei szerint.

3.4. Felületterv

A következőkben bemutatom a rendszer három, várhatóan leggyakrabban látogatott képernyőjét. Az elrendezése mindegyiknek ugyanaz: felül egy menüsáv, ahol a bejelentkezett felhasználó jogosultsági szintjétől függően látszanak a menüpontok.

3.4.1. Kezdőlap

Bejelentkezés után minden felhasználó erre az oldalra érkezik. Áttekintő táblázatot kap az elkövetkező üléseiről, illetve azon jegyzőkönyveiről, melyek már létre lettek hozva, de még nincsenek lezárva. Ennek terve látható a 1. ábrán.



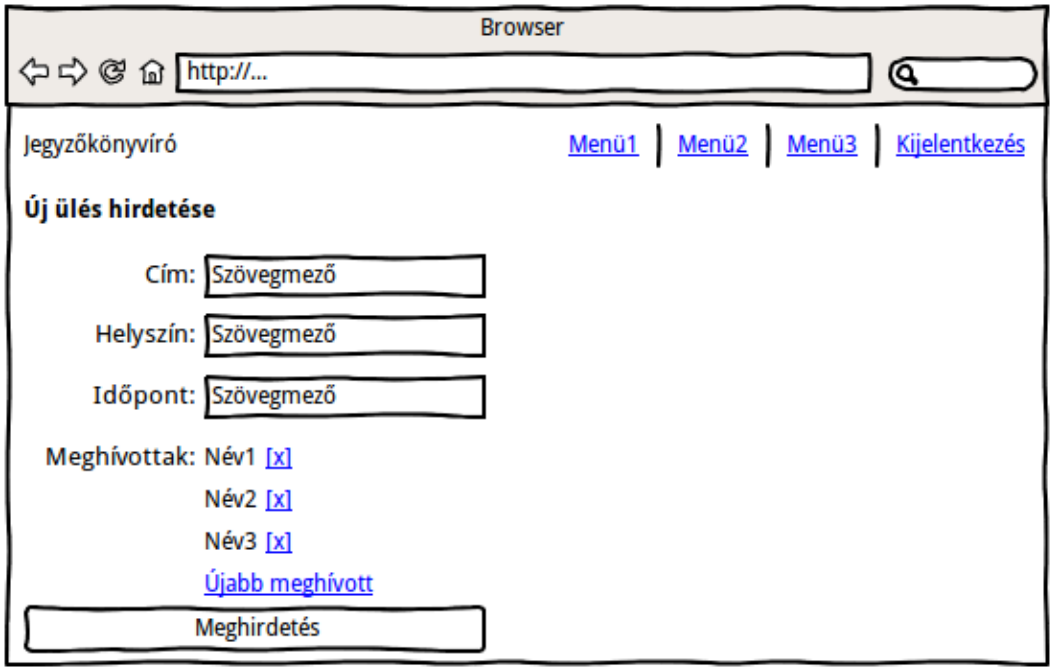
1. ábra. A bejelentkezettek kezdőlapjának látványterve

3.4.2. Üléshirdetés

Ezen a képernyőn lehet megadni az ülés alapadatait és résztvevőket meghívni. Ennek terve látható a 2. ábrán.

3.4.3. Jegyzőkönyvírás

Itt tudja az írással megbízott felhasználó felvinni a jegyzőkönyv elemeit, melyek sorrendfüggően egymás alatt látszanak. Ennek látványterve található a 3. ábrán.



2. ábra. Az ülés hirdetésére szolgáló űrlap látványterve

3.5. Jogosultságok leírása

Néhány mondatban szót ejtek a rendszer által kezelt jogosultságokról, illetve hogy kik lesznek a jellemző tulajdonosai.

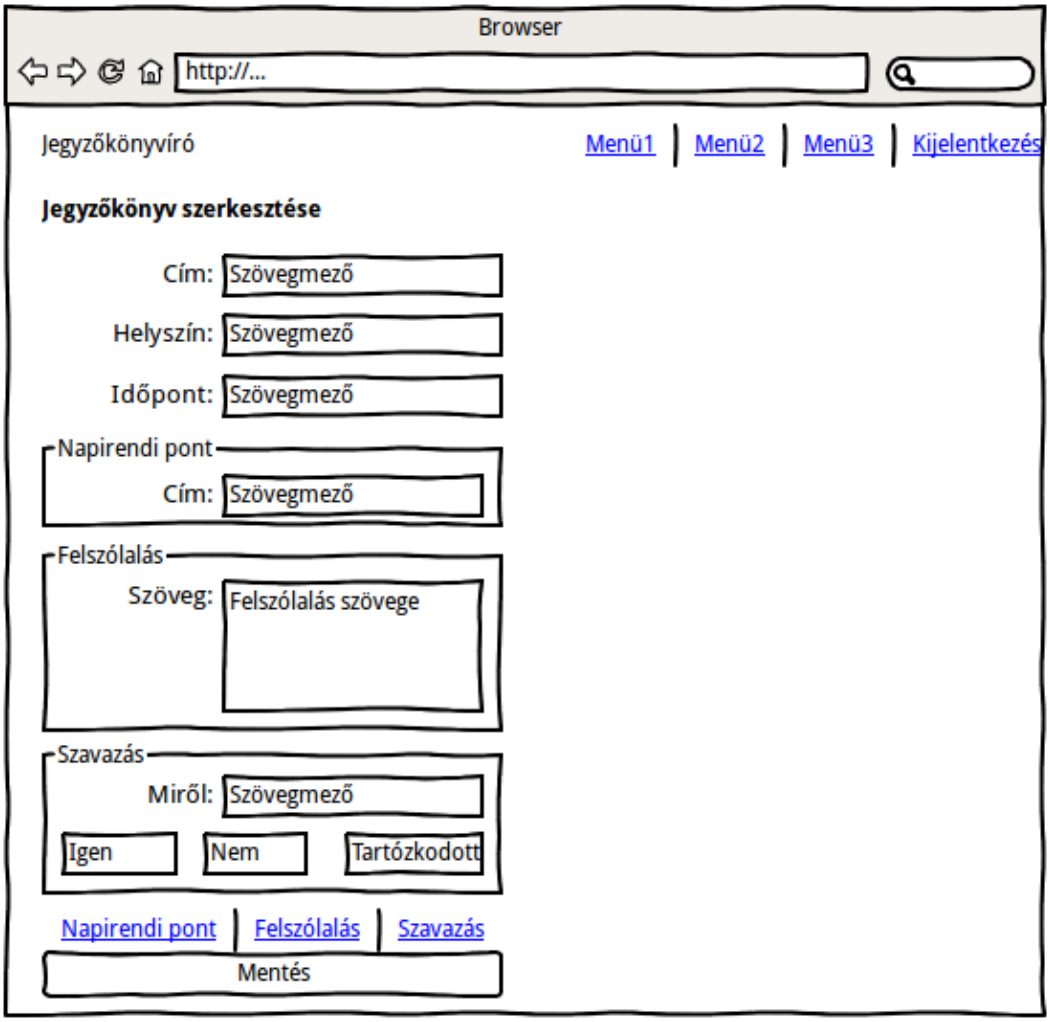
Felhasználó

Az alapszintű jogosultság, hozzáfér a rendszerhez, képes jegyzőkönyveket létrehozni (üléshez párosítva is, illetve önmagában, a rendszeren kívül szervezett ülésekről) és megírni, valamint azon ülések részleteihez és feltöltött anyagaihoz hozzáférni, ahol meghívottként szerepel.

Nem kell külön kiosztani, megkapja mindenki, aki a rendszerhez hozzáféréssel rendelkezik.

Üléshirdető

A fentieken kívül tud még ülést hirdetni, annak jegyzőkönyvírásával megbízni valakit, illetve természetesen az üléseinek adatait szerkeszteni, azokhoz dokumentumokat feltölteni.



3. ábra. A jegyzőkönyv szerkesztésére szolgáló felület látványterve

Jellemzően a kari HÖK elnökök kapják majd ezt a jogot, de rajtuk kívül más személyek is, akik valamely vezető tisztségben vannak.

Adminisztrátor

A fentieken kívül képes a felhasználók kezelésére, adataik megadására (teljes név, szervezeti egység azon belüli pozíció), illetve jogosultságaik kiosztására. Képes tetszőleges ülés adatait és meghívottait módosítani.

Kezdetben egyetlen ilyen jogú felhasználó lesz, a későbbiekben ez az egy igény szerint tud létrehozni újabbakat.

3.6. Jegyzőkönyv létrehozásának folyamata

Egy jegyzőkönyv létrehozása az ülés hirdetésétől kezdve a következő lépésekből áll:

1. *Ülés hirdetése*

   Egy arra jogosult személy meghirdet egy ülést, megadja annak alapadatait (név, helyszín, napirendi pontok), meghívja a résztvevőket.

2. *Ülés jegyzőkönyvírójának megadása*

   Az ülés hirdetője a meghívottak közül kiválasztja a jegyzőkönyvírással megbízott személyt, valamint a jegyzőkönyv két hitelesítőjét.

3. *Jegyzőkönyv alapadatainak megadása*

   A jegyzőkönyv írója kiválaszthatja, hogy melyik ülés jegyzőkönyvét írja, ekkor a rendszer az ismert adatokat átemeli onnan. Alternatívaként létrehozhat jegyzőkönyvet üléshez párosítás nélkül is, azon eseteket kezelendő, mikor az ülés szervezése a rendszeren kívül történt.

4. *Jegyzőköny megírása*

   A megbízott megírja a jegyzőkönyvet tetszőleges számú bejegyzést hozva létre.

5. *Jegyzőkönyv lezárása*

   A jegyzőkönyv írója a folyamat végén lezárja a jegyzőkönyvet, innentől ez nem módosítható.

6. *Jegyzőkönyv hitelesítése*

   A jegyzőkönyv két kijelölt hitelesítője a kezdőlapjukon a megfelelő táblázatban látják, hogy elkészült a jegyzőkönyv. Ezt meg tudják tekinteni, és ha mindent rendben találnak, egy gombnyomással hitelesnek jelölhetik.

   Ha valamelyik hitelesítő hibát talál, erről értesítést tud küldeni a jegyzőkönyv írójának. Ekkor a jegyzőkönyv lezárása is feloldódik, hogy az ismét szerkeszteni tudja.

7. *Jegyzőkönyv elkészülte*

   Mikor mindkét hitelesítő jelezte, hogy a jegyzőkönyv helyes, erről az Ellenőrző Bizottság a beállított e-mail címre értesítést kap, továbbá ha a jegyzőkönyv nyilvános, az publikálásra kerül a rendszer egy bárki számára elérhető oldalán.

Ha a jegyzőkönyv elkészülte után bármilyen okból szükség van erre, a jegyzőkönyvet egy adminisztrátor vissza tudja helyezni szerkeszthető állapotba. Ekkor az írója ismét módosíthat rajta, és a teljes lezárási-hitelesítési folyamatot meg kell ismételni.

A folyamatban két felhasználóra oszlanak a fő feladatok: az ülés hirdetőjére és a jegyzőkönyv írójára. Természetesen a két személy lehet ugyanaz, ha az ülés hirdetője a jegyzőkönyvíró szerepét önmagának osztja ki.

3.7. Tesztelési terv

A Symfony2 remekül támogatatja az egység- és funkcionális tesztek írását. Ezt lehetőségeimhez mérten igyekszem kihasználni, megelőzve azt, hogy a fejlesztés későbbi részeiben valami már működő funkcionalitást elrontsak.

Szintén hasznát fogom venni a Travis-CI szolgáltatásnak, melynek lényege az, hogy minden verziókövetőbe történő commit után automatikusan lefuttatja a teszteket akár több PHP verzión, illetve adatbázis-szerveren.

Emellett amint a rendszer eléri a megfelelő fejlettségi szintet, telepítésre kerül egy alkalmas szerverre, hogy valós felhasználók is elkezdhessék annak kipróbálását és visszajelzéseket küldhessenek.

4. Fejlesztési napló

A rendszer moduljainak fejlesztési sorrendje azok egymásra épülései miatt a következő volt:

1. *Felhasználók*

   Ebben a szakaszban készült el a bejelentkezési folyamat, a felhasználóhoz jogainak hozzárendelése, valamint a felhasználók kezelésére szolgáló rész: azok listázása, új felvétele, létezők szerkesztése.

2. *Ülések*

   Elkészült az üléshirdetési folyamat meghívással, egyelőre a Google Calendar integráció nélkül. Működik a dokumentumfeltöltés, valamint az ülések listázása több szempontból, mint például a saját üléseim, vagy azok, amelyekre meghívtak.

3. *Jegyzőkönyvek*

   Megvalósításra került a jegyzőkönyvek készítésére szolgáló modul. Lehet újat létrehozni csak úgy, illetve üléshez kapcsolva is. A PDF exportálás későbbre marad.

   Ez a szakasz volt a legidőigényesebb.

4. *Tesztelés*

   A rendszer alapvető funkciói elkészültek, így az telepíthető volt egy arra alkalmas szerverre, hogy elkezdődhessen az éles üzemre való felkészítés, illetve hogy valós felhasználói vélemények alapján tudjam a működést finomhangolni.

5. *Naptáresemény és exportálás*

   Megvalósult a Google Naptárral kommunikáló szolgáltatás, hogy a hirdetett ülésekről ott is létrejöjjön esemény, valamint a kész jegyzőkönyvek PDF formátumban történő exportálási lehetősége.

5. A fejlesztés részletei

A következőkben modulokra bontva részletezem a fejlesztés menetét, a főbb lépéseket, a tapasztalt kihívásokat. Előbb azonban ismertetnék néhány fontos koncepciót a Symfony2 felépítéséről és működéséről.

A Bundle rendszer

A symfony 1-es verziója egy monolitikus rendszer volt, nem lehetett egyes komponenseket könnyen cserélni alternatív megoldásokra. Támogatta pluginek használatát, de központi rendszereket ezek sem tudtak helyettesíteni, illetve a nem általános használati esetekben nagyon körülményesek voltak.

A korábbi tapasztalatokra építve a Symfony2-ben új megoldást használtak, ez lett a bundle rendszer[5]. Ez hasonló a pluginekhez, a fontos különbség az, hogy a keretrendszerben *minden* bundle, annak központi funkciói is – ezzel elérték azt, hogy moduláris, könnyen bővíthető rendszert adjanak ki.

Az alkalmazásunk valójában egy minimális, főleg konfigurációból álló ragasztó az azt felépítő bundle-ök körül; ezeket pedig az alkalmazásunk `AppKernel.php` file-jában regisztráljuk:

```
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            // fentebb kihagyva a keretrendszer alapjai,
            // utana a sajatjaink
            new SzakdolgozatSzakdolgozatBundle(),
            new SzakdolgozatFelhasznaloBundle(),
            new SzakdolgozatUlesBundle(),
            new SzakdolgozatJegyzokonyvBundle(),
        );

        return $bundles;
    }
}
```

Ezeket persze lehet feltételesen regisztrálni, azaz megtehetjük, hogy egy adott (mondjuk automatikus naplózáshoz használt) bundle csak az éles szerveren van használva, tesztkörnyezetben nem.

Egy bundle egy adott funkcióhoz szükséges összes file gyűjteménye. Ez lehet néhány alapfunkció is, mint mondjuk néhány adatbázis-tábla leírása és a hozzájuk tartozó osztályok, a másik véglet mondjuk egy egész fórumrendszer. Van néhány általános ajánlás a felosztásra. Semmi nincs kőbe vésve, minden a fejlesztő döntésein múlik.

A Dependency Injection Container

Erre a komponensre nyugodt szívvel lehet mondani, hogy a keretrendszer lelke, az, ami összefogja az egészet. Viszonylag kisméretű (kb. 10 000 sornyi kód), de óriási lehetőségek rejlenek benne.

Ennek a komponensnek segítségével egy helyen tudjuk leírni, milyen szolgáltatásai vannak a bundle-ünknek, azokat milyen azonosítóval lehet elérni, illetve milyen paraméterek szükségesek a példányosításukhoz, helyes működésükhöz. Támogatottak a kötelező függőségek (konstruktoros paraméterátadás) és az opcionálisak is (setteres átadás). A definiált szolgáltatások egésze, vagy akár csak egy-egy paramétere módosítható az alkalmazás szintjén, vagy egy, a mienkre épülő bundle-ben. Paraméterként használhatóak skaláris értékek, de akár egy másik szolgáltatás is.

A szolgáltatások leírása szövegesen, yaml vagy xml formában történhet. Ezek értelmezése lassú folyamat, ami nem megengedhető éles környezetben. Ezen segítendő ezek csak a gyorsítótár ürítése utáni első kéréskor vannak parse-olva, a későbbiek egy PHP-re fordított és mentett, tehát nagyon gyors változatot használnak.

Példaként álljon itt egy szolgáltatás a `SzakdolgozatFelhasznaloBundle`-ből:

```
szakdolgozat.felhasznalo.felhasznalo_param_converter:
    class: Szakdolgozat\FelhasznaloBundle\Request\ParamConverter↩
        \FelhasznaloParamConverter
    arguments: [@doctrine.orm.default_entity_manager]
    tags:
        - { name: "request.param_converter" }
```

Ebben a példában egy úgynevezett ParamConverter szolgáltatást írok le, ezekről a felhasználó modul leírásakor részletesebben lesz szó. Ennek egy konstruktorban átadandó paramétere van, ami egy másik szolgáltatás, a doctrine entity manager-re. Végül adok neki egy adott névvel rendelkező cimkét – amikor a container összeállításra kerül, ezek alapján is lekérdezhetőek a service-ek, és megfelelően használhatóak.

Konfigurációs lehetőségek

Ahogy majdnem mindenre, konfigurációra is több alternatívánk van. Természetesen létezik ajánlás is, amit érdemes követni, ha a bundle-ünket szeretnénk később publikálni.

Az első lehetőség az XML file-ok használata. Nagy előnye, hogy széleskörűen ismert és jól definiált formátum, félreértési lehetőségek nélkül. A symfony.com alatt találhatóak hozzá XSD sémák, ezek segítségével a megfelelő szerkesztőkben kódkiegészítést is kapunk. Hátránya, hogy nagyon bőbeszédű.

A második lehetőség a YAML formátum. Ez szintén egyszerű szöveges formátum, a hierarchia jelölésére szóközökkel való behúzást használ. Például a nem sokkal fentebbi, ParamConverter szolgáltatást regisztráló kódrészlet yaml formátumban van. Előnye, hogy kevés benne a felesleges szöveg, azaz szemmel is könnyen és gyorsan értelmezhető. Hátránya, hogy időnként nem egyértelmű, és a parser hibaüzenetei is gyakran nagyon megtévesztőek.

A harmadik lehetőség az annotációk használata. Más nyelvekben, például javaban ez már régóta nyelvi szinten támogatott, PHP-ban ilyen nincs, segédkönyvtárak alkalmazása szükséges. A Symfonyban a Doctrine Common ezen része használatos, ez a `/**`-al kezdődő kommenteket értelmezi DocBlock-ként. Előnye, hogy a kód és a konfigurációja egy helyen van, hátránya, hogy a nyelv szempontjából kommentben elrejtve, ami egyeseket zavarhat, illetve bizonyos opcode cache-ek ezeket kioptimalizálják, nem működő kódot eredményezve.

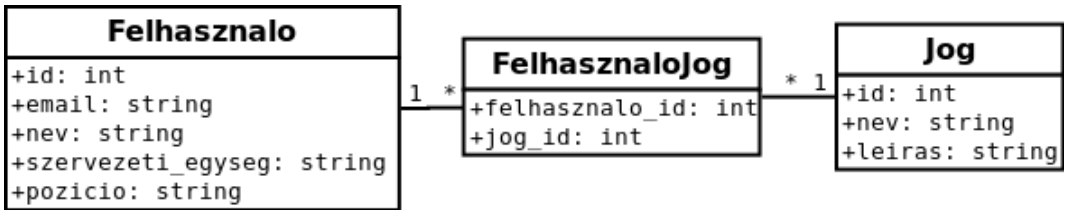
Kutatásom alapján az adatbázis-entitások leírására leginkább használt formátum az annotáció, minden másra a yaml, így a szakdolgozatom fejlesztése során is ezeket használtam.

5.1. Felhasználó modul

Az összes többi modul épít a felhasználókra, így ezt volt logikus elsőnek elkészíteni. A következőkben részletezem a belső felépítését és működését.

Adatbázisának szerkezete

A hozzá tartozó táblák tartalmazzák a felhasználókat, a rendszer által ismert jogokat, illetve hogy ebből melyik felhasználó melyekkel rendelkezik. Szerkezetét lásd a 4. ábrán.



4. ábra. A felhasználó modul adatbázis-szerkezete

A felhasználók OpenID-n keresztüli azonosítása az email címük alapján történik, így ezen a mezőn egyedi megszorítás van. Az id mezőt automatikusan számozza az adatbázis-szerver, ez a tábla elsődleges kulcsa; a felületen megjelenítve sehol nincs, de másik táblák ez alapján hivatkoznak egy bizonyos felhasználóra.

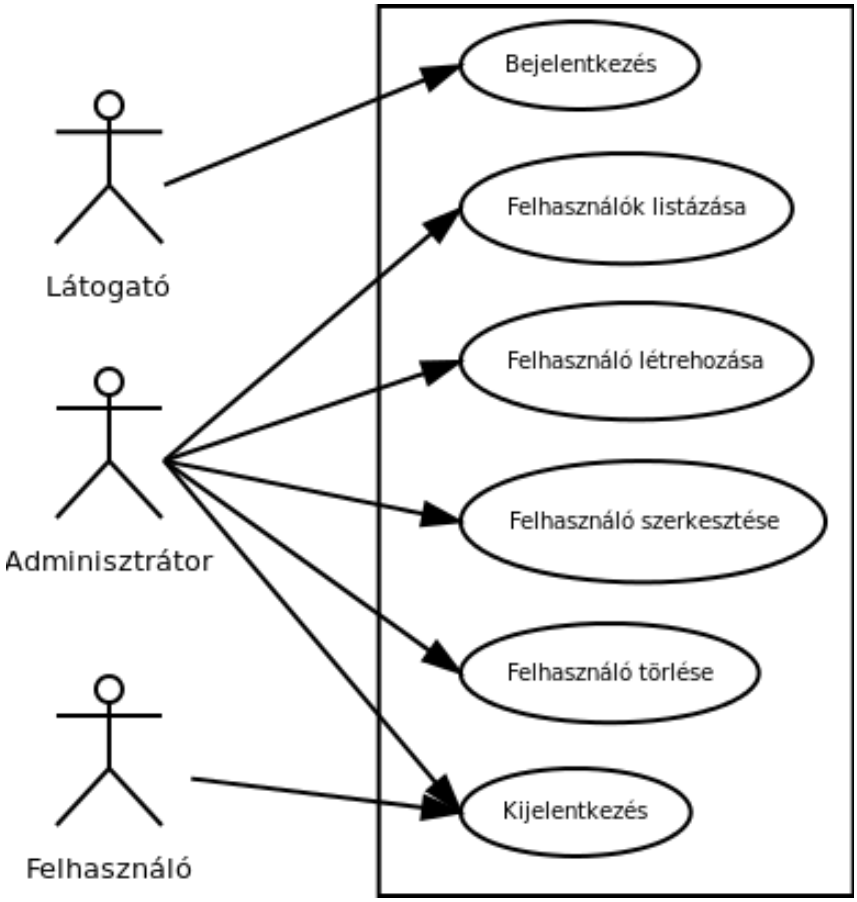
A jog tábla tartalmazza a rendszer által ismert jogokat. A felületen ezek nem szerkeszthetők, az, hogy ezeket nem beégettem a kódba, hanem adatbázisban tárolom, a további fejlesztést teszi könnyebbé. Minden jog rendelkezik egy névvel és egy leírással, felhasználók létrehozásakor és szerkesztésekor ezeket az adminisztrátorok látják.

Az ide tartozó harmadik tábla segédtába, a felhasználó és a jog tábla közötti több-több kapcsolat teszi szükségessé. Egyéb információt nem is tárol, csak (felhasználó, jog) rendezett párokat. A két mező együtt az elsődleges (és így egyedi) kulcs.

Használati esetek

A rendszer használati esetei közé tartozik a felhasználók azonosítása, ennek a folyamatnak része azok jogainak munkamenetbe mentése is; illetve az azok kezelésével kapcsolatos folyamatok: listázás, létrehozás, szerkesztés és törlés, ezek összefoglaló neve a CRUD[6].

A modul felhasználói esetei láthatóak a 5. ábrán.



5. ábra. A felhasználó modul használati esetei

Adatbázis-osztályok

A megvalósítás első lépése az adatbázis tábláinak létrehozása volt. Mivel a Doctrine-nal dolgozom, ezért ezt úgy csináltam, hogy táblánként létrehoztam egy-egy PHP osztályt, és jeleztem a Doctrine-nak, hogy ezek olyan entitások, amiket tárolni szeretnék. Erre annotációkat használtam. Példaként álljon itt a `Felhasznalo` osztály egy részlete:

```
/**
 * @ORM\Entity(repositoryClass="Szakdolgozat\FelhasznaloBundle\↩
    Entity\FelhasznaloRepository")
 * @ORM\Table(name="felhasznalo")
 */
class Felhasznalo implements UserInterface
{
    /**
     * @ORM\Id
```

```
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @ORM\Column(type="string", length=100, unique=true, ↩
        nullable=false)
     */
    protected $email;
}
```

A példában az `ORM`-mel kezdődő annotációkkal foglalkozik a Doctrine. A megvalósított interfész a Symfony2 Security komponensének jelzi, hogy ez az osztály a rendszer felhasználóit tárolja. Erről később írok bővebben.

Az első kettő, az `Entity` és a `Table` jelzi, hogy itt ez egy tárolandó osztály, az adott nevű táblába, és a megadott repository osztállyal – ez az, ahol az egyedi lekéréseinket tudjuk tárolni egy helyen, hogy az könnyen elérhető legyen például a controllerünkből.

Az `id` mezőnél jelezzük, hogy ez azonosító lesz, a mező típusa egész, és hogy az adatbázismotor alapértelmezett generálási sémáját használja – MySQL esetén ez az auto_increment, MSSQL-nél az identity, PgSQL-nél pedig a sequence.

Az email már egyszerűbb, itt csak annyit kell jeleznünk, hogy legfeljebb milyen hosszú karaktersorozatot szándékozunk tárolni, valamint hogy a mező kötelező (azaz nem tartalmazhat null vagy üres értéket), illetve egyedi kell legyen.

Ennek az osztálynak még érdekessége a jogokkal fennálló több-több kapcsolata, ezt az alábbi módon jelzem annotációval:

```
/**
 * @ORM\ManyToMany(targetEntity="Jog", inversedBy="felhasznalok↩
    ")
 * @ORM\JoinTable(name="felhasznalo_jog")
 */
protected $jogok;
```

Azaz egészen egyszerűen csak mondani kell, hogy több-több kapcsolatot szeretnék a `Jog` entity-vel, és a kapcsolótábla neve legyen `felhasznalo_jog`.

Az `inversedBy` mondja meg, hogy a `Jog` entity melyik tulajdonsága ennek a relációnak az ellentéte, abban az osztályban ez a mező így néz ki:

```
/**
 * @ORM\ManyToMany(targetEntity="Felhasznalo", mappedBy="jogok")
 */
protected $felhasznalok;
```

Itt a kapcsolótáblát már nem kell megadnunk, fontos viszont, hogy a korábban látott `inversedBy` helyett `mappedBy` tulajdonságot adunk meg. Ezzel tudjuk jelezni, hogy a kapcsolatnak melyik a tulajdonosi oldala, jelen esetben a felhasználó, azaz hozzá tartoznak a jogok, nem pedig a jogokhoz a felhasználók. Ennek az elemek szerkesztésekor van jelentősége: a Doctrine csak a tulajdonosi oldalt vizsgálja, hogy történt-e változás, kell-e módosítani valamely táblát.

OpenID

A protokoll két felet definiál, a *providert* és a *consumert*. A provider a szolgáltató, ahol a felhasználó rendelkezik már fiókkal, és annak engedélyével megkaphatjuk a kért adatait. A consumer a fogyasztó, jelen esetben a rendszerem, ahová a felhasználó szeretne bejelentkezni a másik oldalnál található fiókjával. A kérés során szabadon megmondhatjuk, hogy a szolgáltatónál elérhető adatok közül melyiket szeretnénk megkapni. Ezek egy része a protokollban van definiálva, de természetesen létre lehet hozni egyedi mezőket is.

A rendszerem tervezésekor még nem tudtam, mennyire összetett ennek a megvalósítása a Symfony keretrendszerben, ezért csak egy szolgáltatóval (Google) történő azonosítást vállaltam. Most már tudom, hogy köszönhetően annak, hogy csak az email mezőre van szükségem, nagyon kevés plusz munkával bármely OpenID kiszolgálóval megvalósítható a bejelentkezés.

A két fél közti kommunikációt a korábban már említett `FpOpenIdBundle` segítségével valósítottam meg.

Bejelentkezés

A rendszer biztonságára az Symfony2 Security komponense ügyel. Először is az alkalmazásom `security.yml` fájljában be kellett állítanom, hogy az egész

rendszerhez OpenID-n keresztüli bejelentkezés szükséges:

```
access_control:
    - { path: ^/login_openid$, roles: ↩
       IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/, role: IS_AUTHENTICATED_OPENID }
```

A szolgáltatótól kapott adatok ellenőrzését a `/login_openid` url-en elérhető controller végzi, így azt el kell tudni érni azonosítás nélkül.

A következő lépés az FpOpenIdBundle konfigurálása volt:

```
fp_openid:
    create_user_if_not_exists: true
    provider: openid_user_manager
    required_attributes:
        - contact/email
```

A `create_user_if_not_exists` beállítás neve megtévesztő, ugyanis nem azt jelenti, hogy hozzon létre felhasználót, ha valaki anélkül próbál bejelentkezni. OpenID-ban egy felhasználó több úgynevezett *identity*-vel rendelkezik, és a beállítás azt mondja meg, hogy ha új identity-vel érkezik egy felhasználó, azzal beengedje-e. A `provider` egy szolgáltatás nevét tartalmazza, amelynek a feladata a providertől kapott adatokból annak eldöntése, hozzáférhet-e a felhasználó a rendszerhez. Az utolsó érték azt mondja meg, mely adatokat szeretnénk megkapni, jelen esetben ez csak az email cím.

A providerként megadott szolgáltatás egy adott interfészt kell megvalósítson, ami egyetlen függvényt tartalmaz. Bemeneti paramétere a szolgáltatótól kapott adatok tömbje, és a végén vissza kell adnia azt a `Felhasznalo` példányt, akit ezek azonosítanak. Ha nincs az adatok közt email cím, vagy ilyennel rendelkező felhasználónk nincs, akkor a függvénynek exceptiont kell dobnia.

Sikeres bejelentkezés után a felhasználó munkamenetébe mentem annak jogait, később ennek függvényében láthat menüpontokat, illetve linkeket. Ezek természetesen szerveroldalon is ellenőrizve vannak, tehát ha egy Adminisztrátor joggal nem rendelkező felhasználó meglátogatja a `/felhasznalo/uj` címet, akkor csak egy hibaoldalt kap.

Felhasználók listázása

A modul funkcióinak fele elkészült, már csak lehetővé kell tennünk, hogy az adminisztrátorok szabadon tudjanak felhasználókat létrehozni, módosítani.

Symfony2-ben a routingot yaml fájlok segítségével tudjuk testreszabni; külön `routing.yml` létezik alkalmazás- és modulszinten. A modul fájljában így írtam le a felhasználókat listázó (index) actiont:

```
szakdolgozat_felhasznalo_felhasznalo_index:
    pattern:    /
    defaults:  { _controller: SzakdolgozatFelhasznaloBundle:↩
       Felhasznalo:index }
```

Az első sor a route neve, a későbbiekben amikor linket generálunk, ezzel tudunk rá hivatkozni úgy, hogy ha a mintát módosítjuk, akkor a linkjeink továbbra is működni fognak. A mintája az adott mappa gyökerére mutat, ezt az egyik következő bekezdésben indoklom. Végül megmondom, hogy ha ez a route illeszkedik a kértre, akkor az aktuális modul `Felhasznalo` controllerének `index` actionjét kell meghívni.

Ettől még nem lesz elérhető ez a route-unk, alkalmazásszintre importálni kell:

```
szakdolgozat_jegyzokonyv:
    resource: "@SzakdolgozatJegyzokonyvBundle/Resources/config/↩
       routing.yml"
    prefix:   /jegyzokonyv
```

Itt leírom, melyik fájlt kell importálni, valamint hogy milyen előtagot kapjon az abban leírt összes útvonal, így a felhasználók listázása a `/felhasznalo/` linken érhető el. Mivel ezt, és a modul ide tartozó összes linkjét csak adminisztrátorok láthatják, a security konfigban védjük is le rögtön:

```
- { path: ^/felhasznalo, role: ROLE_ADMINISZTRATOR }
```

ezt a sort a Bejelentkezés szekcióban mutatott két hozzáférési szabály közé kell írni.

A következő lépés a lista linkelése a menübe. A keretrendszerrel több template nyelv támogatás érkezik beépítve, ebből én a Twiget[14] választottam. Az alapvető vezérlési szerkezetek megtalálhatóak benne, és a Symfony számos függ-

vénnyel kiegészíti, például ezzel, amelyikkel azt tudjuk ellenőrizni, a bejelentkezett felhasználónk rendelkezik-e egy adott joggal:

```
{% if is_granted("ROLE_ADMINISZTRATOR") %}
<li><a href="{{ path('szakdolgozat_felhasznalo_felhasznalo_index↩
    ') }}">Felhasználók</a></li>
{% endif %}
```

Ezután már elkészíthetjük a controllerünk erre az url-re meghívott függvényét. A konvenció szerint a routingban megadott `index` értékre a Symfony a `indexAction` nevű függvényt keresi. Ez a függvény a következőképp néz ki:

```
public function indexAction()
{
    $felhasznalok = $this->getDoctrine()->getRepository("↩
        SzakdolgozatFelhasznaloBundle:Felhasznalo")->findAll();

    return $this->render("SzakdolgozatFelhasznaloBundle:↩
        Felhasznalo:index.html.twig", array(
         "felhasznalok" => $felhasznalok,
    ));
}
```

Azaz lekérjük az összes felhasználót, majd átadjuk a twig template-ünknek. A `findAll` függvény alapértelmezetten az adatbázisbeli sorrendben adja vissza a rekordokat, nekünk ez nem jó, név szerinti ABC-rendezést szeretnénk. Korábban, a `Felhasznalo` entity leírásánál már láthattuk, hogy ez egy saját `Repository` osztállyal rendelkezik, ahol a `findAll` függvényt így definiálom:

```
public function findAll()
{
    return $this->createQueryBuilder("f")
        ->select("f")
        ->orderBy("f.nev", "ASC")
        ->getQuery()
        ->getResult();
}
```

ami az alapértelmezett megvalósítástól egyedül a rendezést leíró sorban tér el.

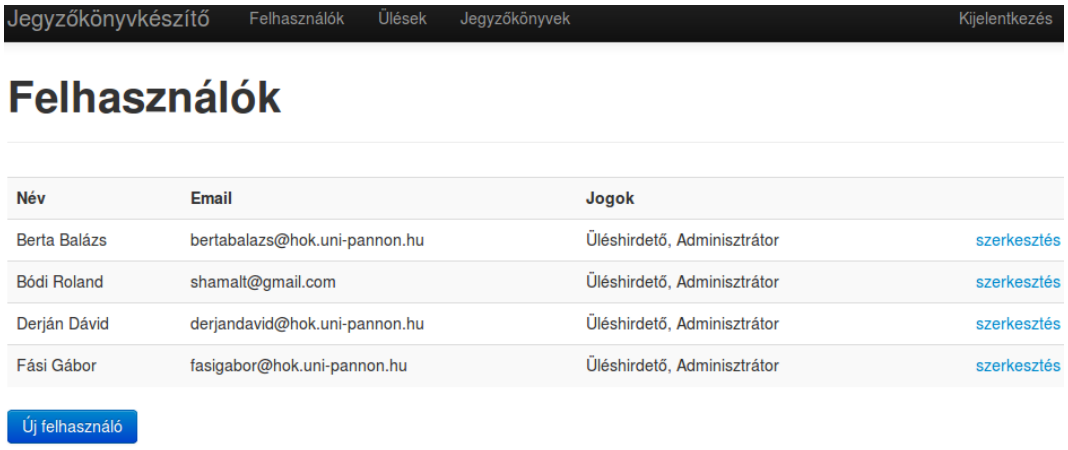
Utolsó lépésként a megfelelő template fájlban végig kell iterálnunk a kapott felhasználókon, kiírni a megjeleníteni szándékozott adataikat, illetve egy szerkesztés

linket:

```
{% for f in felhasznalok %}
<tr>
    <td>{{ f.nev }}</td>
    <td>{{ f.email }}</td>
    <td>{% for j in f.jogok %}{{ j.nev }}{% if not loop.last %},↩
        {% endif %}{% endfor %}</td>
    <td><a href="{{ path("↩
       szakdolgozat_felhasznalo_felhasznalo_edit", { "id": f.id ↩
       }) }}">szerkesztés</a></td>
</tr>
{% endfor %}
```

Ebben a kódrészletben két példát is láthatunk a `for` ciklusra: a külső a felhasználókon megy végig, a belső pedig azok jogain, végeredménye a jogok vesszővel elválasztott névsora lesz. A kimenete látható a 6. ábrán.

A táblázat utolsó cellája egy olyan link lesz, amelyet meglátogatva az a felhasználó szerkeszthető. A következő szakaszban ezt részletezem.



6. ábra. A felhasználók listája

Felhasználók szerkesztése

Most már van linkünk, ami a szerkesztőfelületre mutat, és átadjuk benne a felhasználó azonosítóját. Az action függvényünkben ez alapján lekérjük az adatbázisból és megjelenítjük az űrlapon.

A lekérést megtehetnénk a függvényünk részeként is, de van rá jobb megoldás: egy saját `ParamConverter` írása. Ezek arra jók, hogy az url-ben kapott értékek alapján a függvényünk rögtön egy `Felhasznalo` példányt kaphat. Korábban, amikor a Dependency Injection Containerről írtam, mutattam azt a kódrészletet, amely a felhasználókat id-ből `Felhasznalo` objektummá átalakító `ParamConverter`t regisztrálja.

Ezek az osztályok egy adott interfészt kell megvalósítsanak, amely két függvényt ír le: az első paraméterként egy osztálynevet kap, és vissza kell adnia, támogatja-e az ilyenre átalakítást. A második magát az alakítást végzi paraméterként kapott `Request` alapján. Ebben a példában az url-ben kapott `id` paraméter alapján kérek az adatbázisból, ha nincs ilyen, hibát dobok, amúgy beállítom a `Request`be a konfigurációban beállított kulcs alá. Végeredményként az action függvényünk ennyire leegyszerűsödik:

```
public function editAction(Felhasznalo $felhasznalo, Request ↩
    $request)
{
    $form = $this->createForm(new FelhasznaloType(), ↩
        $felhasznalo);

    return $this->render("SzakdolgozatFelhasznaloBundle:↩
        Felhasznalo:edit.html.twig", array(
         "form"          =>  $form->createView(),
         "felhasznalo"   =>  $felhasznalo,
    ));
}
```

Jól látható, hogy már egy felhasználót kapunk bemeneti paraméterként. Létrehozunk egy formot, melyet feltöltünk az ebben található értékekkel, majd a twiggel rendereltetjük a szerkesztés űrlapot tartalmazó template-et. Abban az űrlap egy részlete így néz ki:

```
<form method="post" action="{{ app.request.requesturi }}">
    {{ form_row(form.nev) }}
    {{ form_row(form.email) }}
```

Azaz az elküldés POST-tal kell történjen, méghozzá az aktuális url-re. A `form_row` twig függvények pedig az űrlap egy-egy sorát renderelik, cimkével,

mezővel és az esetleges hibaüzenettel. Ennek pontos módját testreszabhatjuk, ahogy én is tettem, hogy a Bootstrapnak megfelelő kimenete legyen.

Az elküldött form feldolgozását ugyanaz a függvény végzi, a formot létrehozó, majd a kimenetet renderelő sor között található ez:

```
if ($request->isMethod("post")) {
    $form->bind($request);

    if ($form->isValid()) {
        $this->getDoctrine()->getManager()->flush();

        return $this->redirect($this->generateUrl("↩
            szakdolgozat_felhasznalo_felhasznalo_index"));
    }
}
```

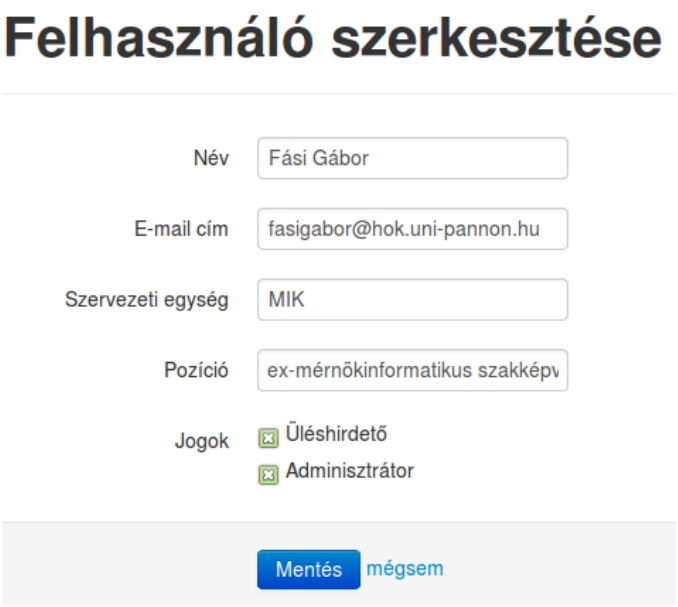
Azaz ha az aktuális kérés POST, akkor a kapott értékeket átadjuk a formnak, ekkor megtörténik azok ellenőrzése. Ha minden érték helyes, akkor a Doctrine Entity Managerén meghívjuk a `flush` függvényt, ami a kezelt objektumok (mint a felhasználónk) módosításait menti az adatbázisba. Végül átirányítjuk a felhasználót a listára.

A form példányunk a Symfony2 Form komponensére épít – itt egy osztályban leírjuk, milyen mezőket szeretnénk és milyen validálási szabályokkal, a controllerünk pedig a fent látható mértékben leegyszerűsödik. A leírást a `buildForm` függvény végzi, egy részlete:

```
public function buildForm(FormBuilderInterface $builder, array ↩
    $options)
{
    $builder->add("email", "email", array(
        "label"         =>  "E-mail cím",
        "constraints"   =>  array(
            new Assert\NotBlank(),
            new Assert\Email(),
        ),
    ));

    $builder->add("jogok", "entity", array(
```

```
            "class"         =>  "SzakdolgozatFelhasznaloBundle:Jog",
            "expanded"      =>  true,
            "multiple"      =>  true,
        ));

}
```

Ebben a példában két mezőt mutatok, az első a felhasználó email címe, amelyhez egy email típusú mező szükséges, valamint kötelező, és szintaktikailag érvényes email cím kell legyen. A második egy adatbázis-entity hivatkozás, méghozzá a Jogra. A fenti beállításokkal egy jelölőnégyzetes listát kapunk, mellyel szabadon meghatározhatjuk, mely jogokkal rendelkezzen a felhasználó. Az űrlap a 7. ábrán látható.



7. ábra. A felhasználók szerkesztésére szolgáló űrlap

5.2. Ülésszervező modul

A fejlesztés logikusan következő lépése az ülések szervezésére szolgáló modul elkészítése volt. Ez már épít a felhasználókra, úgy mint az ülés hirdetője és meghívottjai. Feladata, hogy a létrehozott üléseket tárolja, az azokhoz feltöltött dokumentumokat kezelje és elérhetővé tegye az elérésükre jogosultaknak. Végül értesítenie kell a meghívottakat arról, hogy ülésük lesz.

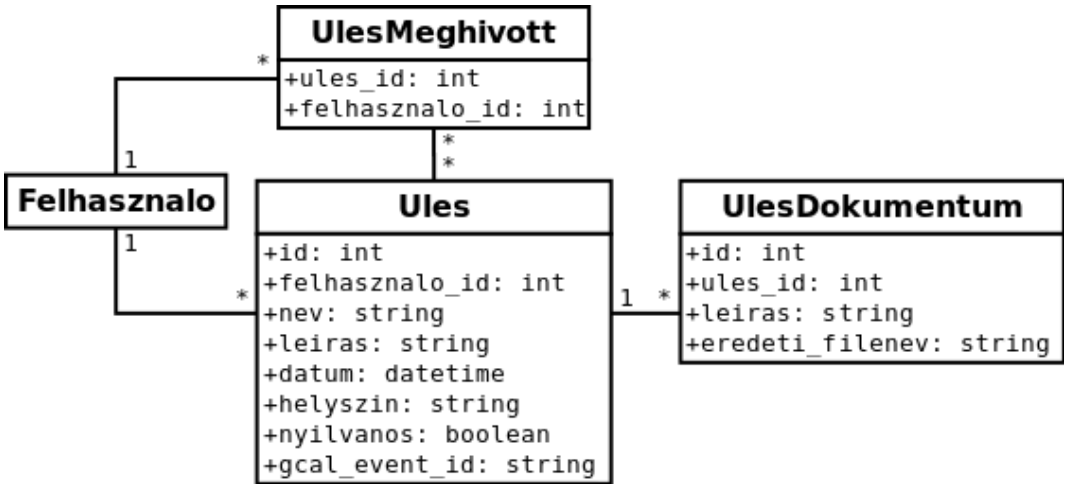
Adatbázisa

Ebben a modulban két entitást kell leírnunk, az egyik az ülés, a másik a dokumentum.

A dokumentumot az azonosítójából generált néven tároljuk a szerveren, egy olyan helyen, ami nem közvetlenül elérhető kívülről, így biztosítani tudjuk, hogy csak a hozzáféréssel rendelkező felhasználók tölthetnek le egy-egy csatolmányt. A generált név miatt azonban tárolnunk kell a file eredeti nevét, hogy letöltéskor így tudjuk kiszolgálni.

Az ülésekről tárolunk néhány alapadatot, ezeken kívül több relációval is rendelkezik: tartozhat hozzá akármennyi dokumentum (természetesen akár nulla is), szintén tetszőleges számú meghívott, valamint pontosan egy hirdető. A meghívottak és a hirdető a felhasználó modulban definiált `Felhasznalo` entitásra hivatkozik.

Az adatbázis szerkezete látható a 8. ábrán. Az `UlesMeghivott` tábla csak az adatbázisban létezik az ülés-meghívott több-több kapcsolat segédtáblájaként, entitás nincs hozzá.



8. ábra. Az ülés modul adatbázisának szerkezete

Korábban nem mutattam még példát egy-több kapcsolatra, mint ami az ülés és a dokumentumok közt van. Az `Ules` osztályban az annotáció a következő:

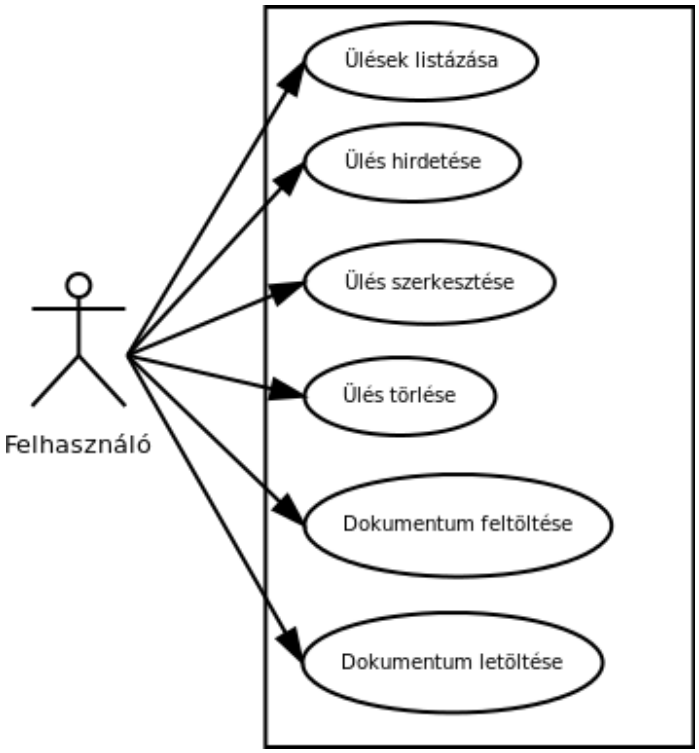
```
/**
 * @ORM\OneToMany(targetEntity="Dokumentum", mappedBy="ules")
 */
protected $dokumentumok;
```

A `Dokumentum` osztályban pedig:

```
/**
 * @ORM\ManyToOne(targetEntity="Ules", inversedBy="dokumentumok↩
    ")
 * @ORM\JoinColumn(name="ules_id", referencedColumnName="id", ↩
    nullable=false)
 */
protected $ules;
```

Használati esetek

Ennek a modulnak a felelőssége az ülések létrehozása, szerkesztése, törlése; azok meghívottjainak és dokumentumainak számontartása; a dokumentumok kiszolgálása jogosultság esetén. Ezek a használati esetek láthatóak a 9. ábrán.



9. ábra. Az ülésszervező modul használati esetei

Új ülés hirdetése

Az üléshirdető joggal rendelkező felhasználók létrehozhatnak új üléseket, megadhatják meghívottjaikat, illetve létrehozás után tölthetnek fel hozzá dokumentumokat.

Az üléslista alján található *Új ülés* gomb a korábban már látott `is_granted` függvénnyel van biztosítva, a controllerünk viszont egy eddig nem látott módszerrel: a megfelelő függvény kapott egy annotációt:

```
/**
 * @Secure(roles="ROLE_ULESHIRDETO")
 */
public function newAction(Request $request)
```

Ezután a Security komponens az ide mutató összes url elérésekor ellenőrzi, a bejelentkezett felhasználó rendelkezik-e ezzel a joggal, ha nem, automatikusan hibaoldalra irányítja.

Az űrlap a korábban már mutatott adatbázis-séma alapadatait tartalmazza, valamint egy listát a rendszer felhasználóiról, mindegyikhez tartozik egy jelölőnégyzet, amivel azt tudjuk megadni: meghívott-e az illető. Ez látható a 10. ábrán. Nagy felhasználószámnál a jelölőnégyzetes megoldás problémás, erre a továbbfejlesztés során szeretnék valami jól használható megoldást találni.

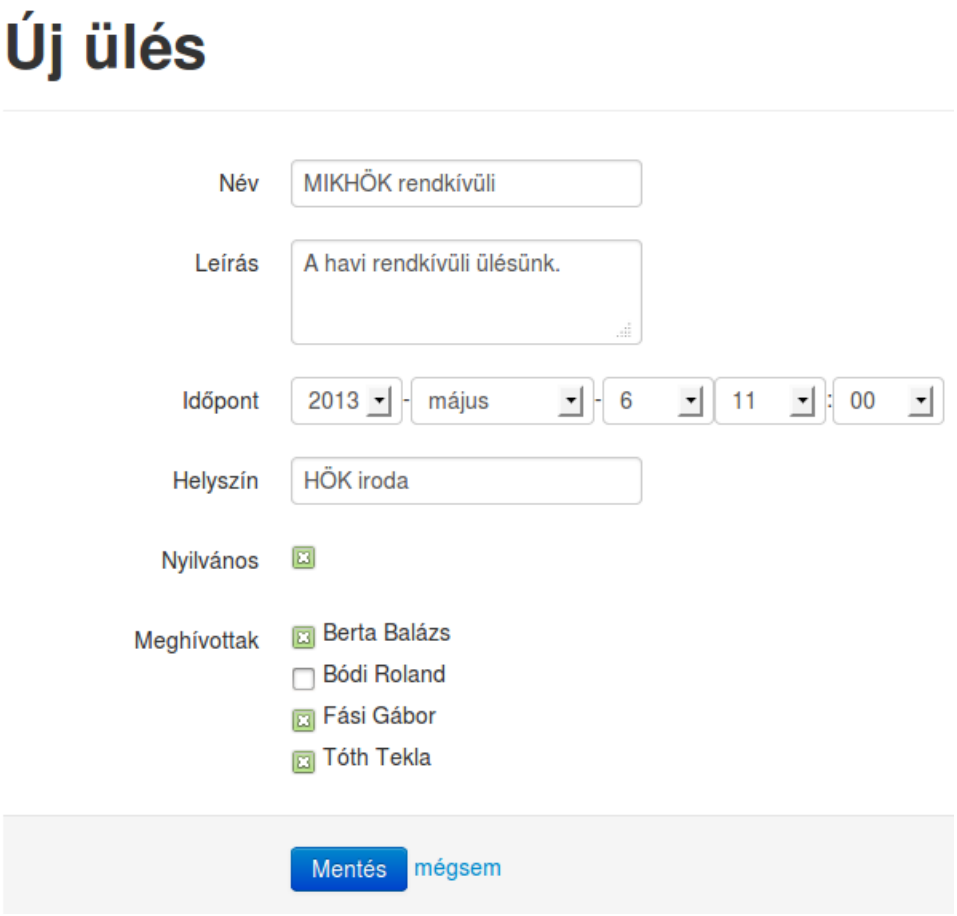
Dokumentumok kezelése

Miután kihirdette az ülést, a szerkesztésére szolgáló oldalon a felhasználó tud hozzá tetszőleges számú dokumentumot feltölteni – ez akkor hasznos, amikor például az ülés egy belső szabályzat tervezett módosításáról szól, ahogy nem sokkal ezelőtt a Hallgatói Önkormányzat tárgyalta a tervezett TVSz-változtatásokat. A dokumentumoknak egy kötelező mezője van, a leírásuk, ide 255 karakter hosszan bármit lehet írni. Tárolom ezen kívül az eredeti fájlnevet, hogy letöltéskor ugyanazon a néven lehessen menteni.

Ezeket a dokumenentumokat később nyilvános ülés esetén bárki, zárt ülés esetén pedig a meghívottak tudják letölteni. A fájlok olyan helyen vannak tárolva, ami nem elérhető kívülről, így a letöltést mindenképp a rendszeren keresztül kell végezni, ahol van lehetőség a jogosultság ellenőrzésére.

A fájlokat kiszolgáló controller egy `ParamConverter` segítségével rögtön egy `Dokumentum` objektumot kap, lérehoz egy választ, beállítja a letöltéshez szükséges fejléceket és a tartalmat, majd visszatér azzal:

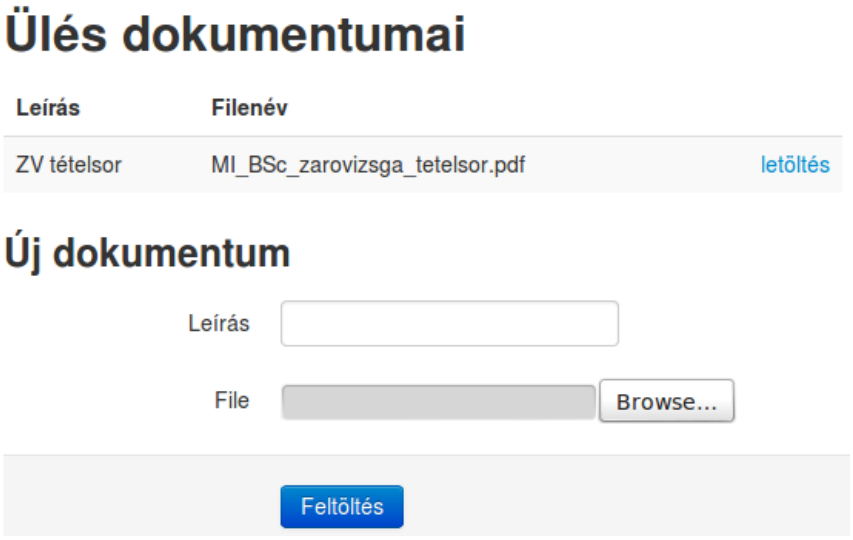
```
public function downloadAction(Dokumentum $dokumentum)
```



10. ábra. Az új ülés hirdetésére szolgáló űrlap

```
{
    $response = new Response();
    $response->setContent(file_get_contents($dokumentum->↩
        getFilename()));
    $response->headers->set("Content-Type", "application/octet-↩
        stream");
    $response->headers->set("Content-Disposition", "attachment; ↩
        filename=" . $dokumentum->getEredetiFilenev());

    return $response;
}
```

Az ülés dokumentumainak kezelérésre szolgáló felület látható a 11. ábrán.



11. ábra. Az ülés dokumentumainak kezelésére szolgáló felület

Google Naptár

A hirdetett ülésekről a rendszer a Google Naptárában létrehoz egy eseményt a rendszerben megadott adatokkal, meghívottakkal. Ennek három fontos előnye van:

1. Esemény létrehozásakor a meghívottak emailben értesítőt kapnak, amiben szerepel az ülés címe, leírása, helye és ideje
2. Esemény módosításakor a meghívottak erről is értesítést kapnak
3. Az esemény a meghívottak naptárában is látszik, ezzel segítve a teendőik átlátását

A kommunikációt a Google hivatalos klienskönyvtárának segítségével valósítottam meg.

A Symfony2 része egy eseménykezelő komponens. Ebben tetszőleges eseményeket hozhatunk létre, és iratkozhatunk fel rájuk. Erre az egyik lehetőség egy *Event Subscriber* osztály készítése, majd regisztrációja:

```
szakdolgozat.ules.gcal_event_subscriber:
    class: Szakdolgozat\UlesBundle\EventSubscriber\↩
       GcalEventSubscriber
    arguments: [@session, @doctrine.orm.default_entity_manager]
    tags:
        - { name: "kernel.event_subscriber" }
```

Egy statikus függvény visszatérési értékével kell meghatároznunk, mely eseményekre akarunk feliratkozni, és azok mely függvényeket hívják meg:

```
public static function getSubscribedEvents()
{
    return array(
        "ules.new"      =>  "ulesNew",
        "ules.edit"     =>  "ulesEdit",
        "ules.delete"   =>  "ulesDelete",
    );
}
```

Definiáltam továbbá egy `GcalEvent` osztályt, az események adataként ilyet továbbítok. Új ülés létrehozásakor a controllerben szükséges kód a következő:

```
$this->get("event_dispatcher")->dispatch("ules.new", new ↩
    GcalEvent($ules));
```

Az `ulesNew` függvényünk gondoskodik a naptárbejegyzés létrehozásáról, valamint annak visszakapott azonosítójának adatbázisba mentéséről – erre később az ülés szerkesztésekor és törlésekor lesz szükségünk:

```
public function ulesNew(GcalEvent $event)
{
    $ules = $event->getUles();

    $event = new \Google_Event();
    $this->ulesAdatokEventbe($ules, $event);

    $service = $this->getGoogleCalendarService();
    $created_event = $service->events->insert($this->calendar_id↩
        , $event, array("sendNotifications" => true));

    $ules->setGcalEventId($created_event->getId());
    $this->entity_manager->flush($ules);
}
```

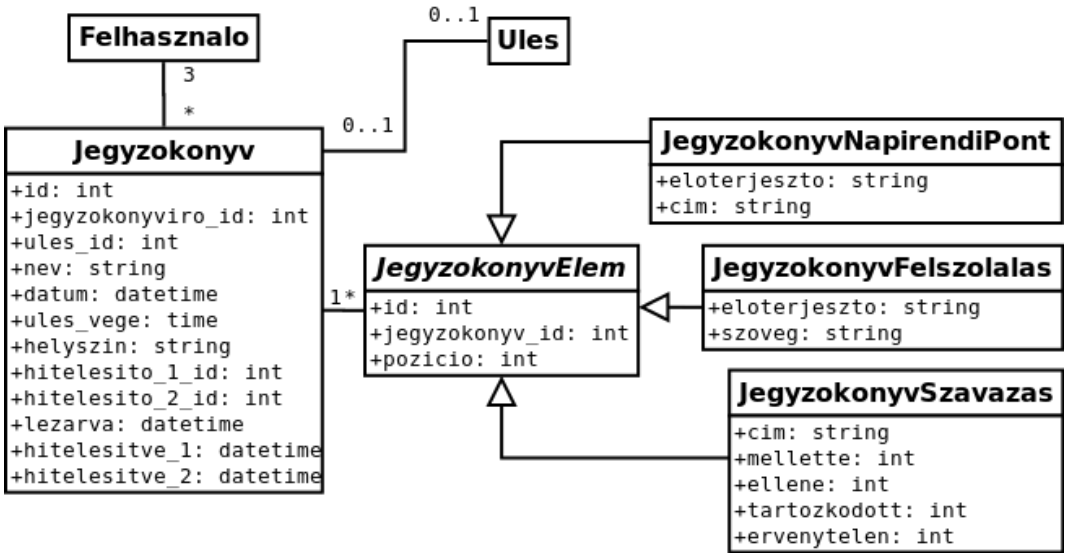
Azaz egy segédfüggvény használatával az ülés adatait átmásolom egy újonnan létrehozott `Google_Event` objektumba, majd egy megfelelően konfigurált `Google_CalendarService` példánynak átadva hozom létre a szerveren a nap-

tárbejegyzést. Végül a létrehozott bejegyzés azonosítóját mentem az ülés adatai közé.

5.3. Jegyzőkönykészítő modul

Adatbázisának szerkezete

Ennek a modulnak a feladata tárolni a jegyzőkönyveket, alapadataikkal és elemeikkel, illetve azok sorrendjével együtt. A szerkezete látható a 12. ábrán.



12. ábra. A jegyzőkönyvkészítő modul adatbázisának szerkezete

A `Jegyzokonyv` tábla két relációval rendelkezik: az egyik a `Felhasznalo` táblára mutat és három mezőre vonatkozik: a jegyzőkönyv írójára és a két hitelesítőre; a másik az `Ules` táblára mutat, és azt jelzi, tartozik-e hozzá ülés. A lezárási, hitelesítési folyamatot dátum mezőkkel ábrázoltam, mert így egyszerre követhető az is, mikor történtek az egyes lépések.

A `JegyzokonyElem` egy absztrakt osztály, a három típusú elem közös mezőit tartalmazza. A Doctrine által biztosított Single Table Inheritance-et használom, ezzel az elemek egy közös táblában tárolódnak plusz egy mezővel, mely megmondja, az adott rekord melyik típusba tartozik. A másik hasonló lehetőség az, amikor elemenként külön táblákban tárolódnak, de mivel az elemek részben közös mezőkkel rendelkeznek, valamint az összes mezőjük kisméretű, a közös táblás tárolás mellett

döntöttem. Bár a tárolás így nem optimális, a lekérdezések sebessége ellensúlyozza ezt.

Használati esetek

E modul feladatai közé tartozik az új jegyzőkönyv létrehozása, ami utána csak a készítője (vagy valamely adminisztrátor) által szerkeszthető. A folyamatban levő vagy kész jegyzőkönyveket bárki megtekintheti, ha a hozzájuk tartozó ülés nyilvános, vagy nincs üléshez párosítva; zárt ülés esetén csak annak meghívottjai. Szintén e modul felügyeli a jegyzőkönyvek lezárási folyamatát, valamint képes azokat PDF formátumban exportálni. Ezeken felül az adminisztrátorok tudják a már létező jegyzőkönyvek készítőjét módosítani. Ezen használati esetek láthatóak a 13. ábrán.

Új jegyzőkönyv létrehozása

A jegyzőkönyvek menüpont mindenki által látható és a felhasználó által látható jegyzőkönyvek listájára mutat. A táblázat alján található gomb az egyik lehetőség az új jegyzőkönyv létrehozására – ekkor az nem kapcsolódik üléshez.

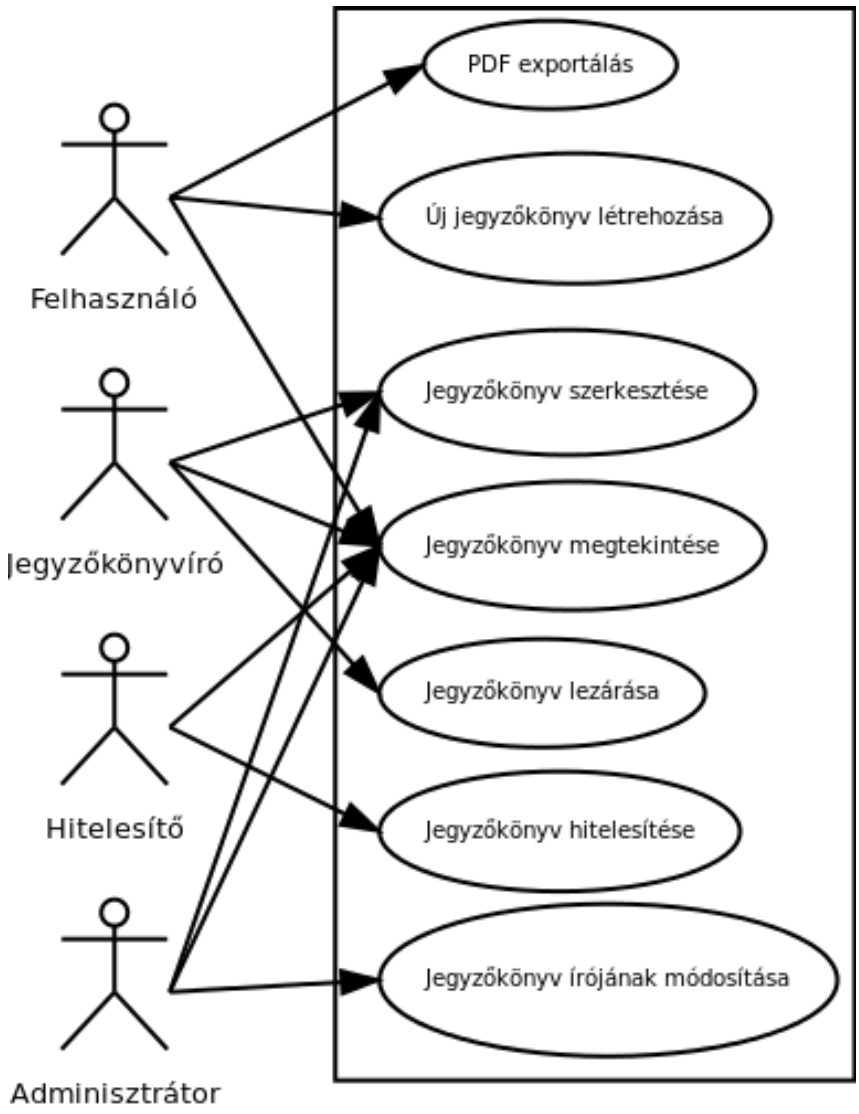
A másik lehetőség a felhasználó kezdőlapján található *Üléseim* táblázat, amibe pontosan akkor kerül a *Jegyzőkönyv létrehozása* link, ha ahhoz az üléshez még nem kapcsolódik jegyzőkönyv.

Mindkét út ugyanarra az űrlapra vezet, ahol először az ülés alapadatait kell megadni. Ha üléshez kapcsolódó jegyzőkönyv készül, akkor ezeket a rendszer automatikusan kitölti, a jegyzőkönyvírónak csak azzal kell foglalkoznia, amit nem tudott, mint például az ülés zárásának időpontja, vagy a jelen levő szavazati jogú tagok száma. Az adatokkal kitöltött űrlap látható a 14. ábrán.

Az alapadatok megadása már elég, hogy a jegyzőkönyvet menteni lehessen.

Új jegyzőkönyv-elemeket létrehozni az űrlap alján található, az egyes típusoknak megfelelő gombokkal lehet; ekkor az új elem a sorrend végére kerül, de a felhasználó egyszerű drag’n’drop módszerrel tetszőleges helyre mozgathatja (ugyanezzel a megoldással átrendezhetőek az elemek a későbbiek során is).

Az egyes elemek vizuálisan jól elkülöníthetőek, ez látható a 15. ábrán. A jobb felső sarkukban található gombbal lehet őket törölni.



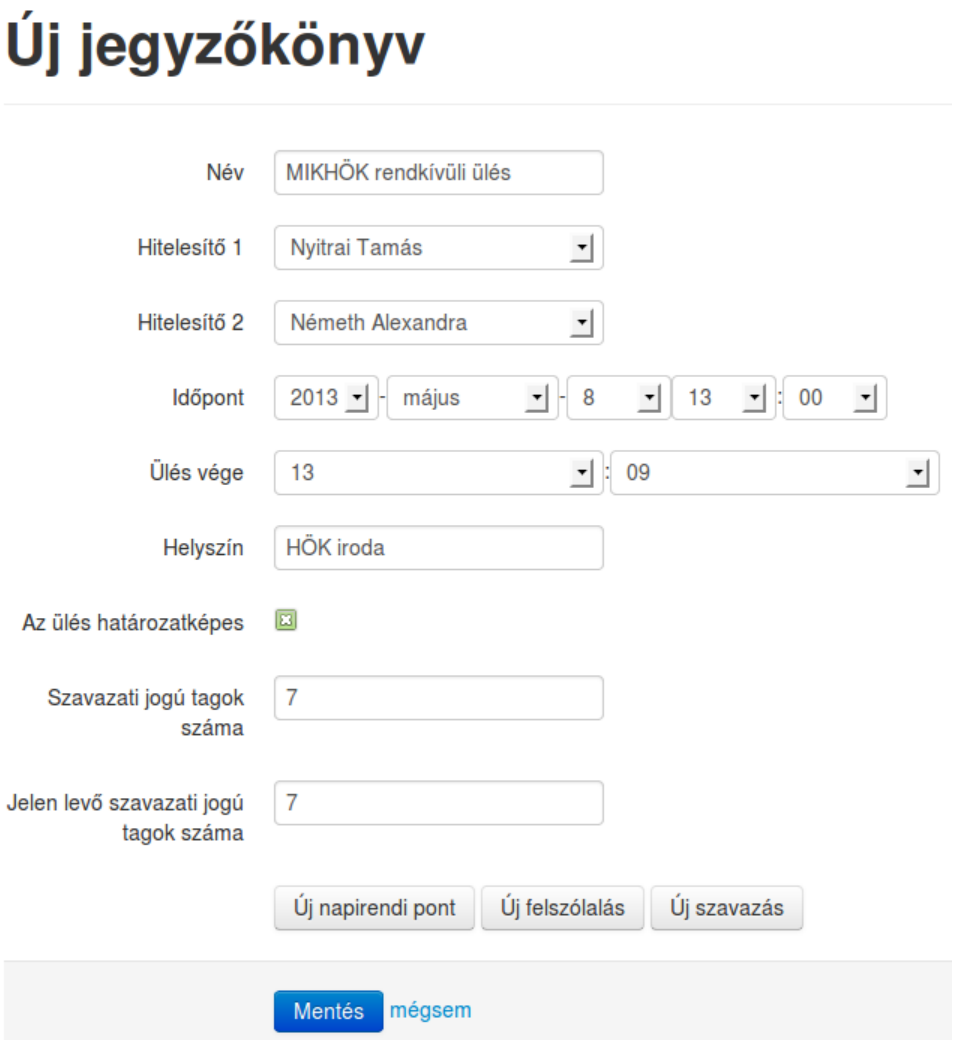
13. ábra. A jegyzőkönyv modul használati esetei

Szerkesztett jegyzőkönyv mentése

Az egész rendszernek a legösszetettebb algoritmusa, ennek a működését szeretném bemutatni. Két adatforrást kell összefésülnie: az adatbázisban levőt, valamint a felhasználótól kapottat.

Amikor a szerkesztés űrlapot elküldi a felhasználó, három tömböt kapok: az első a jegyzőkönyv alapadatait tartalmazza, a második a jegyzőkönyv elemeit, a harmadik pedig a törölt, korábban már adatbázisba mentett elemek azonosítóit. Az elemeket tartalmazó tömb újabb tömböket tartalmaz, amelyekben találhatóak az egyes elemek adatai.

Szerveroldalon a jegyzőkönyvnek, illetve az elemtípusoknak létrehoztam egy-

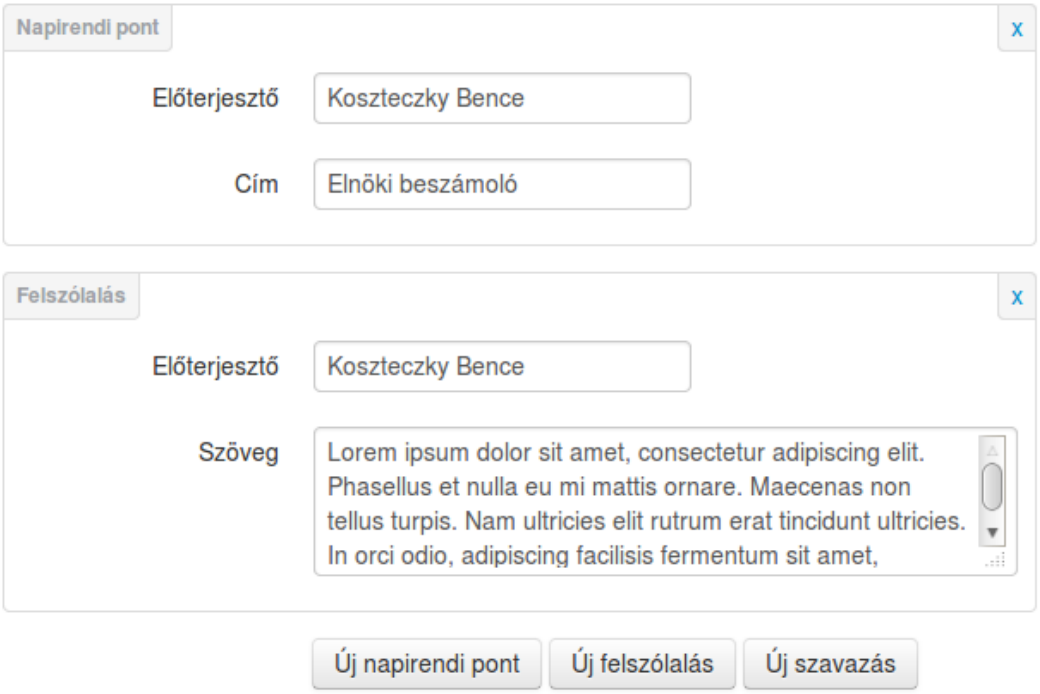


14. ábra. Az új jegyzőköny létrehozására szolgáló űrlap

egy űrlapot a Symfony2 Form komponensében, az azoknak megfelelő mezőkkel illetve validálási szabályokkal. A feldolgozás első lépése az, hogy az adatbázisban található adatokkal és elemekkel létrehozok egy form objektumot a jegyzőkönyvnek és egy-egy darabot az elemeinek.

Az elemekhez tartozó form, illetve a feldogozáshoz szükséges egyéb adatok létrehozását az egyes entitásokra bíztam. A `NapirendiPont` osztályban ez a függvény így néz ki:

```
public function szerkesztesAdatok(FormFactory $factory)
{
    return array(
        "id"    =>  $this->getId(),
        "tipus" =>  "napirendipont",
```



15. ábra. Jegyzőkönyvelemek

```
            "form"   =>  $factory->create(new NapirendiPontType(), ↩
                $this),
        );
    }
```

Ha a kérés GET, akkor innen rögtön a template megjelenítésére ugrom, hisz már megvan mindenem, amire szükségem van hozzá.

A tényleges mentést végző folyamat első lépése az egyes form objektumokhoz az oda tartozó, felhasználótól kapott értékek hozzárendelése. Ennek során az egyes objektumok el tudják dönteni, tartalmaznak-e érvénytelen adatot. A hozzárendelést végző és az elemek érvényességére figyelő kód látható a következő kódrészletben:

```
foreach ($request_elemek as $k => $request_elem) {
    $form = $this->felhasznaloiAdatokbolForm($k, $request_elem);
    $form->bind($request_elem);

    $minden_elem_valid &= $form->isValid();
}
```

Ahol a `$minden_elem_valid` változó kezdeti értéke igaz, végső értéke pedig az összes form validitásának logikai *és* kapcsolata.

Ezután ha az űrlap alapadatai helyesek, és az összes elemet tartalmazó form is helyes, kezdem a kapott adatok adatbázisba tárolását. A jegyzőkönyv már mentve van, itt csak frissíteni kell annak értékeit. Az elemek tömb kulcsának jelentése van:

- ha pozitív, akkor már tárolt elemről van szó, a kulcs az azonosítója, és az adatbázisban módosítanunk kell
- ha negatív, akkor új elemről van szó, és az adatbázisba beillesztenünk kell

A feldolgozást e jelentéseknek megfelelően végzem. A folyamat során figyelek arra is, hogy az elemek sorrendjét helyesen mentsem mind új, mint módosított elemeknél. Az entitások perzisztálását végző ciklus lényegi része a következő:

```
foreach ($elemek as $elem) {
    switch ($elem["tipus"]) {
        case "felszolalas":
            $obj = $elem["id"] < 0
                ? new JegyzokonyvFelszolalas()
                : $this->getDoctrine()->getRepository("↩
                    SzakdolgozatJegyzokonyvBundle:↩
                    JegyzokonyvFelszolalas")->find($elem["id"]);
            break;
        // a többi típust helyszúke miatt nem mutatom
    }

    $data = $elem["form"]->getData();
    if (is_array($data)) $obj->fromArray($data);
    $obj->setPozicio($pozicio++);

    $obj->setJegyzokonyv($jegyzokonyv);
    $em->persist($obj);
    $jegyzokonyv->addElemek($obj);
}
```

Először szükségem van egy entitásra, amely vagy üres, vagy a már adatbázisba mentett rekordot reprezentálja. A `fromArray` függvénnyel hozzárendelem a felhasználótól kapott értékeket, majd beállítom a pozícióját. Végül mindkét irányban beállítom az elem-jegyzőkönyv kapcsolatot, és a Doctrine Entity Managerének jelzem, hogy az objektumot tárolni kell a következő tranzakcióban.

A felhasználótól kapott adatok feldolgozásának utolsó előtti lépése a korábban már adatbázisba mentett, de most törölt elemek eltávolítása. Ezeket egy tömbben kapom meg, melynek minden eleme egy azonosító. A törlést végző függvény látható alant, amelyben természetesen ellenőrzöm, a törölni kívánt elem az épp szerkesztett jegyzőkönyvhöz tartozik-e:

```
protected function toroltElemekTorlese(Jegyzokonyv $jegyzokonyv,↩
     array $toroltelemek)
{
    $repo = $this->getDoctrine()->getRepository("↩
        SzakdolgozatJegyzokonyvBundle:JegyzokonyvElem");

    foreach ($torolt_elemek as $torolt_elem_id) {
        $elem = $repo->find($torolt_elem_id);

        if ($elem->getJegyzokonyv() != $jegyzokonyv) {
            throw new \InvalidArgumentException("Nem a ↩
                szerkesztett jegyzõkönyvbe tartozó elemet próbált↩
                ál törölni");
        }

        $em->remove($elem);
    }
}
```

Sikeres mentés után a felhasználót továbbirányítom a jegyzőkönyvek listájára. Ha a mentés során valahol hibás adattal találkozom, akkor a felhasználó visszakapja a szerkesztés űrlapot az általa megadott adatokkal, jelezve azt, melyik hibás és miért.

Az algoritmust már elkezdtem szétbontani kisebb, egyenként jobban karbantartható és egységtesztelhető függvényekre. A controllerem még így is közel száz sor hosszú, ezt a további fejlesztés során szeretném a töredékére csökkenteni.

PDF exportálás

Számos helyen a mai napig papíralapú iktatást használnak, többek között a Hallgatói Önkormányzat irodájában is. Ezt mindenképpen lehetővé kellett tennie rendszeremnek, így a jegyzőkönyveket lehet PDF formátumban exportálni.

Több PHP nyelvű könyvtár is elérhető, mely ezt teszi lehetővé; a választásom az mPDF-re esett az alábbi okokból:

- ma is aktívan fejlesztik
- jó a dokumentációja
- HTML bemenetből dolgozik

Ebből az utolsó a legfontosabb. Ezzel tudtam azt elérni, hogy a későbbiekben a PDF fájlok tetszés szerint testreszabhatóak legyenek anélkül, hogy a rendszer kódjához hozzá kelljen nyúlni. Csak két érték van beégetve: a papír mérete, valamint hogy álló helyzetű legyen.

Az exportálást végző kódrészletet egy külön osztályba tettem, amelyet szolgáltatásként regisztráltam:

```
szakdolgozat.jegyzokonyv.export.pdf:
    class: Szakdolgozat\JegyzokonyvBundle\Export\Pdf
    arguments: [@templating]
```

Látható, hogy paraméterként várja a keretrendszer templating szolgáltatását, a twiget. Ennek segítségével a jegyzőkönyvek készítése során használhatóak az abban található ellenőrzések, vezérlési szerkezetek, így akár azt is lehetővé téve, hogy ha nem adminisztrátor exportál, akkor valamilyen vízjelet kapjanak az oldalak.

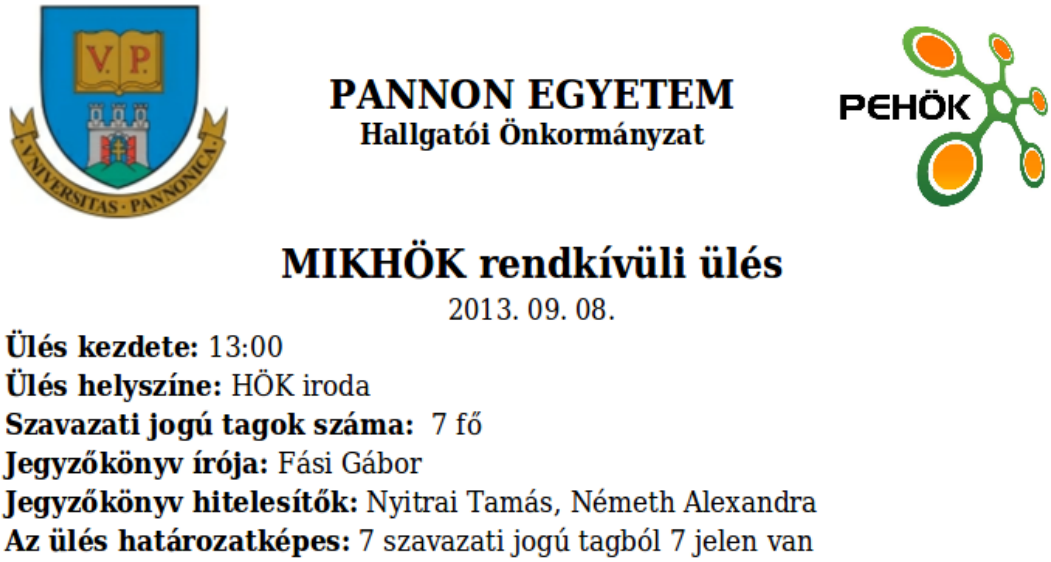
Az osztályunkban a függvény mindössze pár soros:

```
public function jegyzokonyv(Jegyzokonyv $jegyzokonv)
{
    $mpdf = $this->alapMPDF();

    $mpdf->WriteHTML($this->templating->render("↩
       SzakdolgozatJegyzokonyvBundle:Pdf:jegyzokonyv.html.twig",↩
        array("jegyzokonyv"   =>  $jegyzokonv)));

    $mpdf->Output($jegyzokonv->getNev() . ".pdf", "I");
}
```

Az első sor egy mPDF objektumot hoz létre, mely az alapbeállításokat tartalmazza: a lap mérete, orientációja, valamint hogy mely fájlokban találhatóak a fej-

és láblécek leírásai. A második sor ennek beállítja a HTML tartalmát arra, amit a twig generál, majd a harmadik sor visszaadja azt a felhasználónak.

A fejlesztés során elkészítettem egy olyan kimenetet, amely nagyon hasonló ahhoz, amilyet a Hallgatói Önkormányzat most használt rendszere készít, ennek egy részlete látható a 16. ábrán.



16. ábra. Az exportált jegyzőkönyv egy részlete

5.4. Általános modul

Van egy maréknyi oldal, amely több modul szolgáltatásaival is dolgozik, de valójában egyikbe sem illik; a legfontosabb ilyen a bejelentkezettek kezdőoldala, ahol üléseket, jegyzőkönyveket listázok különböző szűrési feltételek alapján. Ezeket egy külön, `SzakdolgozatBundle` nevű modulba gyűjtöttem össze. A következőkben bemutatom, hogy az említett kezdőlapon milyen listák találhatóak.

A fontossága miatt az első azokat az üléseket listázza, amelyek még nem kezdődtek el, és vagy nyilvánosak, vagy a felhasználó hirdeti, vagy pedig meghívott. Ezek szűrésére a Doctrine `QueryBuilder` osztályát használtam az `UlesRepository` osztály `kovetkezoUleseim` függvényben, mely az ülés entitáshoz kapcsolódó repositoryban található:

```
public function kovetkezoUleseim($felhasznalo)
{
    $qb = $this->createQueryBuilder("u");
```

```
    return $qb
        ->select("u")
        ->innerJoin("u.meghivottak", "m")
        ->where("u.datum >= ?1")
        ->andWhere(
            $qb->expr()->orX(
                $qb->expr()->eq("u.nyilvanos", "1"),
                $qb->expr()->eq("u.felhasznalo", "?2"),
                $qb->expr()->eq("m", "?3")
            )
        )
        ->orderBy("u.datum", "ASC")
        ->setParameter(1, new \DateTime())
        ->setParameter(2, $felhasznalo)
        ->setParameter(3, $felhasznalo)
        ->getQuery()
        ->getResult()
        ;
}
```

Sajnos a `QueryBuilder` nem teszi lehetővé, hogy reláció alapján szűrjek, ezért kellett ahhoz a cselhez folyamodnom, hogy összekapcsolom az üléseket a meghívottjaikkal, és az így kapott plusz mezők között szűröm a felhasználót.

A második táblázatban találhatóak azok a jegyzőkönyvek, amelyeket a felhasználó hozott létre, és még nincsenek lezárva. Ez a következőképpen van szűrve:

```
public function lezaratlanJegyzokonyveim($felhasznalo)
{
    $qb = $this->createQueryBuilder("j");

    return $qb
        ->select("j")
        ->where($qb->expr()->isNull("j.lezarva"))
        ->andWhere($qb->expr()->eq("j.jegyzokonyviro", "?1"))
        ->setParameter(1, $felhasznalo)
        ->getQuery()
        ->getResult()
        ;
```

```
}
```

A harmadik táblázatban vannak azok a jegyzőkönyvek, melyek már le vannak zárva, a felhasználó az egyik hitelesítőjük, és még nem hitelesítette őket. A szűrést végző függvény:

```
public function hitelesitesreVaroJegyzokonyveim($felhasznalo)
{
    $qb = $this->createQueryBuilder("j");

    return $qb
        ->select("j")
        ->where($qb->expr()->isNotNull("j.lezarva"))
        ->andWhere(
            $qb->expr()->orX(
                $qb->expr()->andX(
                    $qb->expr()->isNull("j.hitelesitve_1"),
                    $qb->expr()->eq("j.hitelesito_1", "?1")
                ),
                $qb->expr()->andX(
                    $qb->expr()->isNull("j.hitelesitve_2"),
                    $qb->expr()->eq("j.hitelesito_2", "?2")
                )
            )
        )
        ->setParameter("1", $felhasznalo)
        ->setParameter("2", $felhasznalo)
        ->getQuery()
        ->getResult()
        ;
}
```

A negyedik táblázat azokat az üléseket listázza, amelyek a közelmúltban történtek, és nyilvánosak, a felhasználó hirdette őket, vagy meghívott volt; valamint még nem készült hozzájuk jegyzőkönyv. Az ebben a táblázatban található linkkel lehet ezt lérehozni, a rendszer automatikusan elvégzi az összekapcsolást.

Az üléseket kigyűjtő függvény csak a szűrésekben tér el az előzőtől:

```
->where("u.datum <= ?1")
->andWhere("u.datum >= ?2")
```

```
->andWhere(
    $qb->expr()->orX(
        $qb->expr()->eq("u.nyilvanos", "1"),
        $qb->expr()->eq("u.felhasznalo", "?3"),
        $qb->expr()->eq("m", "?4")
    )
)
->orderBy("u.datum", "ASC")
->setParameter(1, new \DateTime())
->setParameter(2, $mult)
->setParameter(3, $felhasznalo)
->setParameter(4, $felhasznalo)
```

A `$mult` változó egy olyan `DateTime` objektum, amely a három héttel ezelőtti dátumot tartalmazza.

A kódrészleteken jól látható, milyen egyszerűen lehet megvalósítani összetett lekérdezéseket is a Doctrine segítségével.

6. Az elkészült munka értékelése

Dolgozatom témája egy olyan összetett rendszer elkészítése volt, melyben önmagában is számos nehéz pont található, de a legnagyobb kihívást a külső rendszerekkel való kommunikáció jelentette; mint a Google Naptár események kezelése, PDF exportálás és az OpenID authentikáció. Ezeket részben készen elérhető komponensek segítségével, de jól sikerült megoldanom.

Rendszerem készítése során számos új technológia használatát sajátítottam el, és ezek segítségével olyan alkalmazást sikerült készítenem, amely megfelel a felhasználók igényeinek, emellett a kódja is könnyen karbantartható, továbbfejleszthető; ezekre különös figyelmet fordítottam, hisz a munka nem ér véget a szakdolgozatom leadásával.

Bár a kezdeti lépések a számomra idegen környezet, keretrendszer miatt nehezek voltak, számos olyan megoldását tettem magamévá, amelyek a későbbiekben nagyon megkönnyítették a dolgomat; a két legfontosabb ilyen az adatbázis-entitásokat kezelő Doctrine, valamint a Symfony2 Form komponens használatának elsajátítása volt.

A tervezés, fejlesztés során folyamatosan egyeztettem a Hallgatói Önkormányzat Ellenőrző Bizottságával az igényeikkel kapcsolatban, ennek megfelelően fejlődött a rendszer; amit április közepe óta egy tucat felhasználó tesztel. Számos remek visszajelzést kaptam már tőlük; a kezdőlap az ő kéréseik alapján készült el, valamint ők vetették fel a több jegyzőkönyvtípus kezelésének lehetőségét is. Összességében arra számíhatok, hogy megelégedéssel fogják használni.

7. Továbbfejlesztési lehetőségek

Van néhány olyan feladat, amiket nem tartalmaz a vállalt témám, de a jövőben szeretném megvalósítani. Néhány ilyet már jeleztem korábban, azokon túl az alábbiak a legfontosabbak.

Az első ilyen a keretrendszer frissítése. A fejlesztést a 2.1-es verzióban kezdtem, és annak során adták ki a 2.2-es verziót számos olyan változtatással, amelyek visszafelé nem kompatibilisek. Ezeket szeretném tanulmányozni, és a programom úgy módosítani, hogy ne okozzanak gondot. Ezen túl május végére várható a 2.3-as verzió, amely az első LTS, azaz hosszútávon támogatott változat lesz – ez itt három évet jelent.

A második ilyen újabb authentikációs módok hozzáadása. Jelenleg csak Google fiókkal lehet belépni, de ezen belül nincs korlátozás. Viszonylag kevés munkával meg lehet oldani egyéb, OpenID-kompatibilis szolgáltatókkal való belépést, valamivel több munkával pedig a hagyományos, felhasználónév-jelszó használatával történőt is.

A harmadik ilyen a mobileszközök használhatóságának vizsgálata. Egy mai mobiltelefon kijelzője túl kicsi, hogy hosszabb szöveg bevitelét tegye lehetővé, de a terjedő tabletek már elég nagyok. A megjelenítésre használt Bootstrap keretrendszer ezeket valamennyire támogatja, de rendelkezésemre álló eszköz híján ezt tesztelni nem tudtam.

Az utolsó ilyen a jegyzőkönyvtípusok támogatása. Jelenleg csak egyet támogat a rendszer, de már felmerült az igény, hogy többet is tegyen lehetővé. Ennek hatása csak az exportált PDF kinézetére terjed, a létrehozási, szerkesztési és lezárási folyamat minden esetben ugyanaz lenne.

8. Irodalomjegyzék

[1] POTENCIER, F., ZANINOTTO, F. (2010) *A Gentle Introduction to symfony 1.4* Sensio Labs

[2] WAGE, J., BORSCHEL, R., BLANCO, G. (2010) *Doctrine ORM for PHP* Sensio Labs

[3] http://www.albacomp.hu/index.php?pg=menu_8715 (letöltés dátuma 2013. május 1.) *Albacomp Testületi Munkát Támogató Rendszer*

[4] https://openid.net/specs/openid-authentication-2_0.html (letöltés dátuma 2013. május 1.) *az OpenID protokoll specifikációja*

[5] http://symfony.com/doc/current/book/page_creation.html#the-bundle-system (letöltés dátuma 2013. május 1.) *Creating Pages in Symfony2, The Bundle System*

[6] https://en.wikipedia.org/wiki/Create,_read,_ update_and_delete (letöltés dátuma 2013. május 1.) *Create, read, update and delete*

[7] http://symfony.com/blog/dailymotion-powered-by-symfony *Dailymotion, powered by symfony*

[8] http://www.doctrine-project.org/ (letöltés dátuma 2013. május 1.) *Doctrine Project*

[9] http://www.ekozig.hu/dt.html (letöltés dátuma 2013. május 1.) *eKÖZIG döntéstámogató rendszer*

[10] http://intermap.hu/termekek/termekekonkormanyzat.html (letöltés dátuma 2013. május 1.) *InterMap e-FORTE*

[11] http://knpbundles.com/formapro/FpOpenIdBundle (letöltés dátuma 2013. május 1.) *FpOpenIdBundle leírása*

[12] https://code.google.com/p/google-api-php-client/ (letöltés dátuma 2013. május 1.) *Google API PHP kliens*

[13] https://developers.google.com/google-apps/calendar/v3/reference/events (letöltés dátuma 2013. május 1.) *Google Calendar Event API*

[14] http://twig.sensiolabs.org/ (letöltés dátuma 2013. május 1.) *Twig, The flexible, fast, and secure template engine for PHP*

[15] http://twitter.github.com/bootstrap/ (letöltés dátuma 2013. május 1.) *Twitter Bootstrap*

[16] http://symfony.com/blog/yahoo-answers-powered-by-symfony (letöltés dátuma 2013. május 1.) *Yahoo! Answers powered by symfony*

9. Mellékletek

A szakdolgozat CD mellékletének könyvtárszerkezete:

/FasiGabor-GFIVNG-szakdolgozat.pdf

/szakdolgozat-forraskod

- /diagramok
- /kepek
- /wireframek
- /szakdolgozat.tex

/rendszer-forraskod

- /app
- /src
  - /google-api-php-client
  - /Szakdolgozat
    - FelhasznaloBundle
    - JegyzokonyvBundle
    - SzakdolgozatBundle
    - UlesBundle

/uploads

/web

/internetes-hivatkozasok

/albacomp_testuleti_munkat_tamogato_rendszer

/az_openid_protokoll_specifikacioja

/create_read_update_and_delete

/creating_pages_in_symfony2_the_bundle_system

/dailymotion_powered_by_symfony

/doctrine_project

/ekozig_dontestamogato_rendszer

/fpopenidbundle_leirasa

/google_api_php_kliens

/google_calendar_event_api

/intermap_eforte

/twig_the_flexible_fast_and_secure_template_engine_for_php

/twitter_bootstrap

/yahoo_answers_powered_by_symfony